UNIVERSITÀ DI PISA

Facoltà di Scienze Matematiche Fisiche e Naturali

Tesi di Laurea Magistrale in Tecnologie Informatiche

Un algoritmo di marker tracking basato su Local Descriptors e Image Matching per applicazioni di Realtà Aumentata.

Relatori

Prof. Franco Tecchia

Prof. Marcello Carrozzino

Candidato

Felice Giovinazzo

Anno Accademico 2010/2011

Dedica

Melissa

Papà

Mamma

Giuseppe

Introduzione

La presente tesi affronta il tema della *marker detection*, ovvero l'uso di tecniche di computer vision per la stima del posizionamento di marker noti all'interno di un flusso di immagini catturate attraverso una telecamera, e si pone l'obiettivo di sviluppare algoritmi in grado di riconoscere marker composti da immagini naturali, in alternativa alla ferrea codifica bidimensionale usualmente utilizzata dagli approcci esistenti. Il lavoro basa le proprie fondamenta su un'estesa analisi dello stato dell'arte dei principali algoritmi esistenti di *natural features detection*, di cui analizza nel dettaglio le prestazioni sia in termini di velocità che di affidabilità del processo di detection. Sulla base di tale studio viene poi presentata un'architettura in grado di impiegare features naturali nel task di riconoscimento dei marker stessi, con l'inserimento dell'approccio sviluppato all'interno di una libreria open-source per la marker detection. Le prestazioni del sistema così risultante vengono poi misurate ed analizzate, facendo poi un'analisi critica dell'utilizzabilità pratica dell'approccio proposto per applicazioni di Realtà Aumentata.

Struttura della tesi

Si da alla seguente tesi un'organizzazione in 7 capitoli principali:

Il **Capitolo** 1 fornisce una definizione di Realtà Aumentata descrivendo

e collocando al suo interno il problema fondamentale dell'Image Matching.

Il **Capitolo** 2 da una descrizione dello stato dell'arte degli algoritmi per Image Matching off-line.

Nel **Capitolo** 3 viene descritto lo stato dell'arte di altri algoritmi per Image Matching con caratteristiche tali da poter permettere calcoli veloci e l'impiego in applicazioni interattive.

Il **Capitolo** 4 fornisce la descrizione di un tipico algoritmo di marker detection con marker fiduciali bianchi e neri e introduce ArUco, una libreria opensource per la Realtà Aumentata.

Il **Capitolo** 5 presenta l'implementazione di un tracker di marker con caratterestiche naturali basato su ArUco.

Nel **Capitolo** 6 vengono presentati e commentati i risultati di alcuni test effettuati allo scopo di fornire una valutazione delle prestazioni di alcuni tra gli algoritmi descritti nello stato dell'arte e impiegati nel processo di tracking implementato in questo progetto.

Infine il **Capitolo** 7 espone le conclusioni finali e i possibili sviluppi futuri del presente lavoro.

Indice

Elenco delle figure

Capitolo 1

Introduzione alla Realtà Aumentata

La Realtà Aumentata *(AR: Augmented Reality)* è una tecnologia che consiste nell' integrare gli elementi che appaiono nella realtà con elementi virtuali generati da un computer (a differenza della realtà virtuale che genera un ambiente completamente fittizio). Questa tecnologia permette di arricchire le immagini reali con un insieme potenzialmente infinito di informazioni aggiuntive (per esempio testi, fotografie, modelli tridimensionali, ecc...) e contemporaneamente all'utente di non perdere il contatto con l'ambiente reale circostante. Nei sistemi di realtà virtuale (VR) l'utente viene completamente *immerso* in un ambiente fittizio, interattivo e tridimensionale generato dal computer. La tipica interfaccia di un sistema di realtà virtuale è l'HMD (Head Mounted Display) che permette la visione di uno scenario virtuale tridimensionale, separando completamente l'utente dalla realtà circostante, impedendone quindi il movimento all'interno di un ambiente reale, ma costringendolo a usufruire della sola realtà virtuale. Nei sistemi AR, invece, immagini o altri dati virtuali vengono uniti con l'ambiente percepito dall'u-

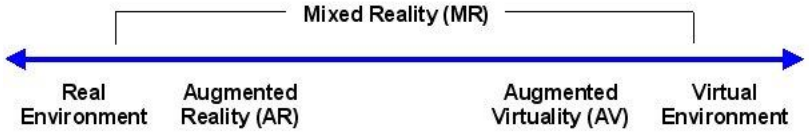


Fig. 1.1: Mixed Reality Continuum.

tente e se le due fonti di informazione (la vista dell' utente e le immagini virtuali) sono ben coordinate, l'effetto che si ottiene è quello di una sovrapposizione coerente che permette all'utente che fa uso del sistema di usufruire delle informazioni aggiuntive generate dalla macchina percependole strettamente correlate con la realtà esterna e con i propri sensi. In letteratura esistono principalmente due definizioni di Realtà Aumentata: La prima è data da Milgram, che ha proposto una classificazione per identificare le relazioni sussistenti tra Realtà Aumentata e realtà virtuale. Egli ha definito un *mixed reality continuum* (1.1, pagina 14), rappresentato da una retta ai cui estremi giacciono il mondo reale percepito dai nostri sensi e la Realtà Virtuale completamente fittizia dei sistemi VR. Tutti i punti intermedi di questa retta fanno parte della mixed reality dove gli elementi reali e virtuali sono entrambi presenti, in misura maggiore o minore, a seconda che ci si avvicini ad un estremo oppure all'altro. In questo continuum la Realtà Aumentata occupa la parte che più si avvicina alla realtà. Quando gli elementi virtuali prevalgono su quelli reali si parla di Augmented Virtuality. La classificazione formulata da Milgram si basa su tre caratteristiche che un sistema può avere:

- **Conoscenza del mondo**: è la prima condizione da soddisfare nella costruzione di un'applicazione AR, occorre una approfondita conoscenza sia del mondo reale che di quello virtuale che si vogliono unire, in modo da ottenere un corretto allineamento tra elementi virtuali e reali. A questo scopo sono necessarie procedure di calibrazione e tracking per determinare cosa deve essere visto dall'utente a seconda della sua

posizione nel mondo reale e del suo punto di vista. Questa procedura in alcune applicazioni può essere relativamente semplice (generalmente in tutti quei casi in cui l'ambiente reale è noto in tutti i suoi dettagli, come nel caso di ambienti interni), in altre applicazioni invece diventa più complessa (quando non è disponibile una conoscenza completa del mondo reale: per esempio in ambienti esterni).

- **Accuratezza della riproduzione**: è la qualità con cui le immagini virtuali appaiono all'utente e il loro grado di coerenza con l'ambiente reale. Desiderabile sarebbe l'avere immagini sintetiche fotorealistiche ben integrate con la realtà, tanto da ottenere una realtà mista nella quale le immagini virtuali siano indistinguibili da quelle reali, tuttavia il costo computazionale necessario per ottenere questo risultato è da ritenersi ancora troppo elevato per i stringenti vincoli temporali a cui si e' soggetti nella grafica interattiva.
- **Sensazione di presenza**: questa proprietà è più critica nei sistemi VR, in quanto l'utente deve essere in grado di interagire con l'ambiente virtuale ed orientarsi in esso, quindi la sensazione di presenza dipende in gran parte dal tipo di display utilizzato. Nei sistemi AR, invece, tale sensazione è di norma presente dato che l'utente si muove ed interagisce in un ambiente reale arricchito con elementi virtuali.

La seconda definizione di AR è stata formulata da R. Azuma nel 1997, il quale afferma che un sistema AR deve avere le seguenti caratteristiche:

- **Combinare elementi reali ed elementi virtuali**, è la caratteristica fondamentale di un sistema AR.
- **Interattivo in tempo reale**: il sistema deve reagire e aggiornarsi in tempo reale a seconda delle azioni dell'utente.
- **Oggetti reali e virtuali devono essere ben integrati in un am-**

biente tridimensionale: è la caratteristica che distingue un sistema AR dagli altri sistemi di mixed reality. Le immagini virtuali devono essere geometricamente allineate agli elementi del mondo reale.

1.1 Applicazioni

La Realtà Aumentata ha trovato impiego nei più disparati campi: si va dalle applicazioni in ambito militare a quelle per uso medico, dalle applicazioni ingegneristiche a quelle per il controllo remoto di macchinari e per il lavoro di assemblaggio in fabbrica, dalle applicazioni a scopo di intrattenimento a quelle realizzate in ambito educativo-culturale (guide per musei, siti archeologici, ecc...). In campo militare, per esempio, la Realtà Aumentata viene da tempo impiegata nella strumentazione dei velivoli militari, per dare informazioni aggiuntive al pilota senza distrarlo dagli eventi esterni. Uno dei campi più importanti è quello medico, dove possono essere utilizzate immagini tridimensionali sovrapposte alla vista reale del corpo del paziente, in modo tale da aiutare i medici nella diagnosi di malattie o nelle operazioni chirurgiche. La AR può essere utilizzata nel controllo remoto di apparecchiature robotiche, mostrando all'operatore il punto di vista del robot con una vista tridimensionale sulla scena. Troviamo applicazioni anche nelle operazioni di assemblaggio o riparazione di componenti: al tecnico vengono mostrate direttamente sullo schermo le informazioni dettagliate delle operazioni che deve svolgere, magari fornendo anche un modello 3D dettagliato del componente, e guidandolo passo passo durante il lavoro. In ambito ingegneristico esistono applicazioni AR che permettono a più utenti che si trovano in località differenti di collaborare ad uno stesso progetto, permettendo a tutti di osservare uno stesso oggetto virtuale. Nel campo dell'intrattenimento basta pensare al

largo utilizzo che viene fatto di studi televisivi virtuali, dove, con il metodo *chroma-key*, attori reali, ripresi su uno sfondo blu, appaiono allo spettatore posizionati in una scena virtuale. Per quanto riguarda l'ambito culturale, esistono numerose applicazioni fornite a musei ed altre istituzioni culturali, che permettono al semplice visitatore oppure allo studioso, di ottenere maggiori e più dettagliate informazioni sulla sua visita. Questo può avvenire fornendo contributi audio-visivi riguardanti le opere o i siti archeologici che l'utente sta osservando, oppure fornendo ricostruzioni realistiche di edifici storici in rovina oppure che sono andati perduti.

1.2 Architettura di un sistema AR

Il sistema AR è frutto di diverse tecnologie, tra le quali l'elettronica, l'informatica, la meccanica e la robotica. Dal punto di vista tecnico, un'architettura AR consiste, in genere, di quattro componenti principali: display, dispositivo di tracking, dispositivi per l'acquisizione delle immagini (videocamere, webcam, ecc...) e un computer che generi la grafica virtuale (deve essere in grado di generare le immagini in modo che appaiano all'utente in tempo reale).

- Display: è la componente hardware che si frappone tra l'utente e la realtà che lo circonda, la sua funzione è quella di mostrare all'osservatore immagini virtuali allineate in maniera consistente rispetto all'ambiente reale. I display possono essere posizionati in maniera differente rispetto all'utente:
  - Near eye / Head attached display: sono posizionati vicino all'occhio dell'utente e quindi vanno indossati da chi osserva, per esempio lenti oppure caschi (HMD: Head Mounted Display).
  - Hand held display: vengono tenuti tra le mani dell'osservatore,

per esempio uno smartphone oppure un palmare.

  - Spatial display: sono collocati nello spazio tra l'osservatore e l'oggetto osservato, per esempio pannelli, schermi, ecc...

  Nella scelta del display vanno considerati quattro importanti fattori:

  - Latenza: intervallo di tempo che intercorre tra il movimento dell'utente e l'istante in cui l'immagine viene visualizzata dopo che viene elaborata.
  - Risoluzione della scena e distorsione: il modo in cui la scena viene presentata all'osservatore e eventuali distorsioni provocate dall'ottica impiegata.
  - Campo visivo: porzione di vista permessa dal display all'utente.
  - Fattori di costo: dipendono dalla complessità delle ottiche utilizzate.

- Tracking: è necessario per ottenere la posizione e l'orientamento dell'osservatore rispetto ad un sistema di riferimento, in modo da ottenere il corretto allineamento delle immagini virtuali con quelle reali. Se l'utente cambia punto di vista, il sistema di tracking deve fare in modo che l'immagine artificiale rimanga allineata con l'oggetto che viene osservato.

  Attualmente esistono diversi sistemi di tracking:

  - Meccanici: misurano la posizione utilizzando una connessione meccanica diretta tra un punto di riferimento e l'utente
  - Elettromagnetici: si basano sulla misura del campo elettromagnetico generato da appositi dispositivi indossati dall'utente.
  - Acustici: utilizzano onde sonore ad alta frequenza per misurare posizione e orientamento degli oggetti.
  - Inerziali: si basano su accelerometri e giroscopi che permettono di

calcolare posizione e orientamento degli oggetti in base alla loro posizione e velocità iniziali.

- Ottici: sono utilizzati per tracciare posizione e orientamento del punto di vista dell'utente. Si basano su due componenti: sorgenti di luce e sensori ottici per il riconoscimento della posizione degli oggetti. Il sistema più comune è l'utilizzo di videocamere per rilevare degli appositi *fiducial* posti nella zona in cui si vuole far comparire l'oggetto virtuale, ed in base ai quali l'applicazione AR è in grado di dedurre il punto di vista dell'utente. Si tratta del sistema sul quale si basa il presente lavoro di tesi.
- Ibridi: utilizzano tecniche miste per sfruttare i vantaggi che offre ciascun approccio.

1.2.1 Funzionamento

Il sistema AR ha come primo obiettivo quello di riuscire ad integrare, nella maniera migliore possibile, le immagini virtuali e reali. Mentre le immagini virtuali vengono in ogni caso generate dal computer, per quanto riguarda le immagini reali, possono essere trattate in modi diversi. Il primo metodo consiste nel registrare tramite una videocamera le immagini reali dal punto di vista dell'utente, i frame vengono quindi analizzati in tempo reale per determinare la posizione in cui devono apparire gli oggetti virtuali e per identificare quali oggetti reali di interesse compaiono nell'ambiente. Questa fase di *tracking* è importante perchè da essa dipende la corretta visualizzazione degli oggetti virtuali. Un modo per farlo è tener traccia dei movimenti e dell'orientamento della videocamera rispetto ad un punto di riferimento noto (come, per esempio, un fiducial) (vedi figura 1.2). Una volta determinate posizione e orientamento degli oggetti, il computer fonde le immagini provenienti dalla

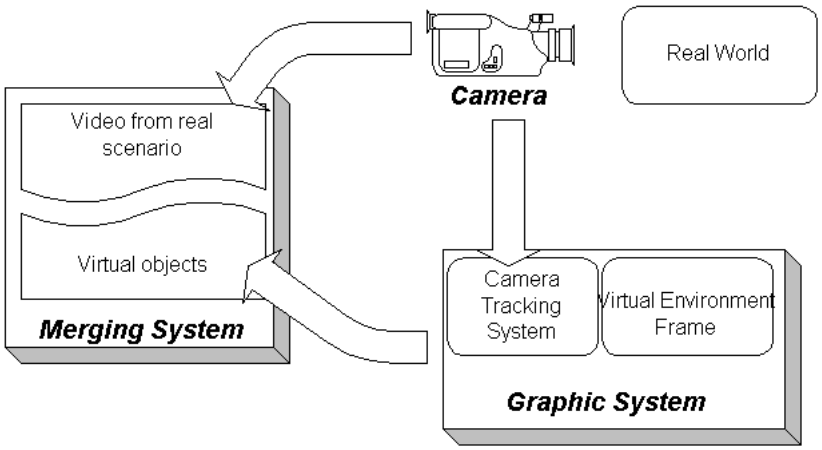


Fig. 1.2: Sistema AR con tracking sulla videocamera.

videocamera con quelle virtuali mostrando il tutto sul display dell'utente. Il secondo metodo consiste nell'utilizzare un display separato (per esempio lenti di cui è possibile regolare l'opacità per permettere all'utente di vedere direttamente il mondo reale) (vedi figura 1.3).

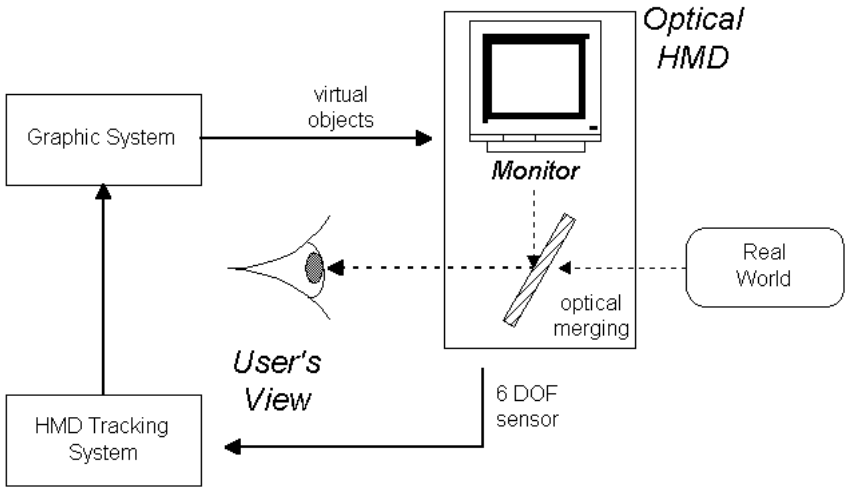


Fig. 1.3: Sistema AR con HMD optical see-through.

Questo sistema permette l'osservazione diretta dell'ambiente reale, mentre per posizionare ed orientare correttamente gli oggetti virtuali è sufficiente

tracciare (per esempio tramite un sensore $6DOF$[1])i movimenti della testa dell'utente. Il metodo offre inoltre la possibilità di rilevare occlusioni tra il mondo reale e gli oggetti virtuali (oggetti reali che si sovrappongono a quelli virtuali) e trattarle di conseguenza. In ogni caso è indispensabile l'utilizzo di una videocamera per trattare i due fenomeni nel caso in cui ci siano oggetti in movimento, per l'analisi del campo visivo e per conoscere in tempo reale quali modifiche subisce lo scenario in cui l'utente si sta muovendo (per esempio se l'utente muove una mano nel suo campo visivo, va trattata l'occlusione che si verifica tra la mano ed eventuali oggetti virtuali che si trovano dietro). Il display HMD (Head Mounted Display) costituisce il tipo

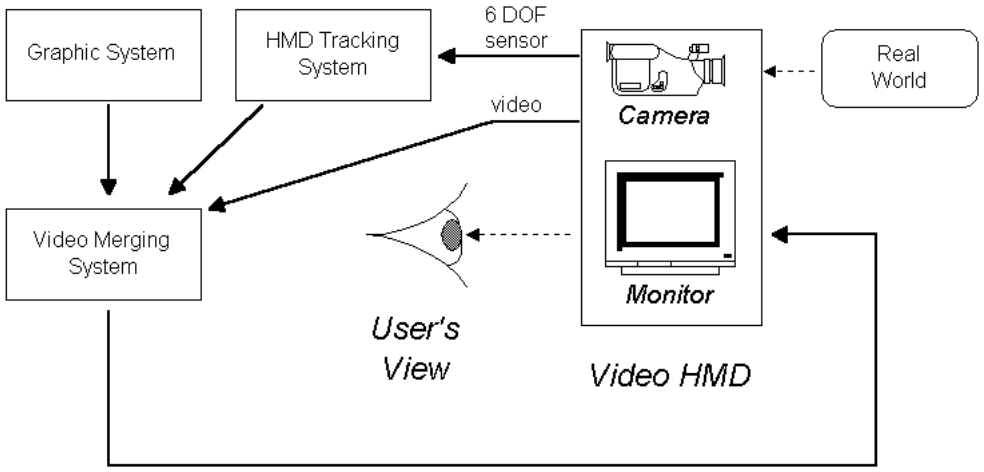


Fig. 1.4: Sistema AR con HMD video see-through.

di display ottimale per queste applicazioni. Permette di isolare la visuale dell'utente dall'ambiente esterno. Questi display sono fondamentalmente di due tipi: *video see-through* (fig. 1.4) e *optical see-through* (fig. 1.3). Le differenze principali tra i due tipi di display consistono nel modo in cui l'utente osserva l'ambiente esterno. Come già precedentemente accennato, nel

[1]6 Degrees Of Freedom: 6 gradi di libertà, ovvero un sensore capace di rilevare movimenti lungo i sei assi principali.

caso del *video see-through* l'utente osserva l'ambiente esterno indirettamente, tramite dei display che mostrano le immagini della videocamera già elaborate (aumentate), questo comporta diversi problemi, tra i quali il fatto che a causa della latenza nella visualizzazione delle immagini ci possono essere difficoltà nel movimento dell'utente. Nell'*optical see-through*, invece, l'utente percepisce l'ambiente esterno osservandolo direttamente attraverso delle lenti e gli oggetti virtuali vengono mostrati con un sistema ottico basato su lenti ad opacità variabile. Questo metodo supera i problemi del *video see-through*: l'utente osserva l'ambiente in maniera più naturale ed è in grado di muoversi più liberamente e con maggiore sicurezza. Va considerata infine la latenza, ovvero il ritardo con cui le immagini arrivano al computer dalla videocamera. Inoltre il sistema ha bisogno di ulteriore tempo per elaborare le immagini virtuali in maniera coerente ai dati di tracking del sensore, tale intervallo di tempo sarà tanto più grande quanto più sono numerose e complesse le immagini virtuali da elaborare. Una latenza eccessiva può causare il venir meno della coerenza tra le immagini virtuali e quelle reali, compromettendo così il corretto funzionamento del sistema AR.

1.3 Image Matching nell'AR

Nell' Augmented Reality gioca un ruolo molto importante il riconoscimento delle immagini all'interno, per esempio, di un flusso video. Un problema comune in questo ambito è quello dell' *object detection*, ovvero il riconoscimento di oggetti all'interno di una scena reale e complessa, seguito dall'eventuale sovrapposizione di uno strato virtuale(*AR layer*) che evidenzi e descriva i caratteri individuati. L'*image matching*, cioè il confronto tra immagini che rappresentano lo stesso oggetto inquadrato da differenti punti di vista, è un

aspetto fondamentale in molti problemi della Realtà Aumentata, come il riconoscimento di oggetti, la ricostruzione di ambienti tridimensionali partendo da sequenze di immagini che ritraggono l'ambiente da diverse angolazioni, il *motion tracking*, ecc. Nell' immagine Fig. 1.5(pagina 23) si descrive il flusso

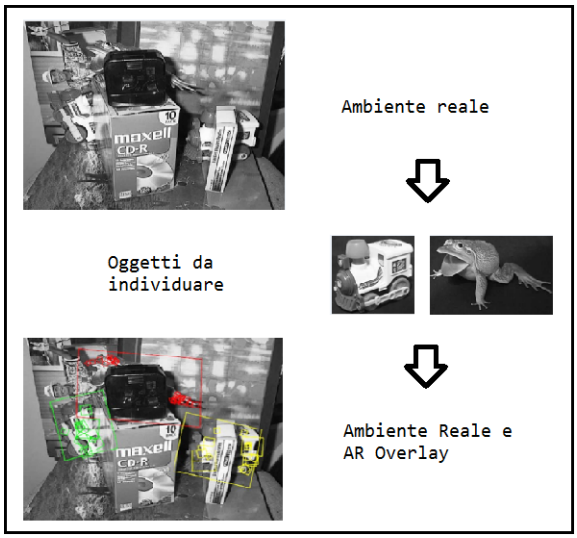


Fig. 1.5: Passi principali di algoritmi di object detection.

standard di un algoritmo di object detection:

- fase preliminare nella quale vengono calcolate le caratteristiche principali degli oggetti da riconoscere.
- applicazione di algoritmi specifici per l'estrazione e la descrizione di punti caratteristici all'interno delle immagini.
- riconoscimento attraverso l'image matching degli oggetti.
- sovrapposizione overlay che evidenzi e localizzi gli oggetti all'interno della scena reale.

1.4 Tracking marker-based e Tracking markerless basato su features naturali

Nell'ambito della Realtà Aumentata e ai fini di questo lavoro è utile distinguere due tipologie di tracking ottico,il primo utilizza immagini bidimensionali chiamate comunemente *marker*, mentre l'altro si basa sull'analisi locale dell'immagine con lo scopo di individuare oggetti con catteristiche naturali(differenti dai classici marker bitonali). [20].

1.4.1 Tracking marker-based

Nella Computer Vision il tracking basato su marker si fonda sul riconoscimento, all'interno di una scena, di alcuni pattern caratteristici facilmente individuabili in base al forte contrasto dei bordi(solitamente quadrati o rotondi di colore nero con sfondo bianco) rispetto al resto della scena. In base alla rappresentazione dei bordi e della figura all'interno dei bordi si possono classificare i marker in differenti sottogruppi. Solitamente il riconoscimento dei bordi all'interno della scena viene ottenuto distinguendo le zone della scena in cui i gradienti dei pixel passano, drasticamente, da un valore ad un altro(passaggio da colori di scena a bordo del marker). La discriminazione tra un marker ed un altro viene effettuata in base ad una codifica del valore racchiuso tra i bordi individuati.

Si possono distinguere varie tipologie di marker:

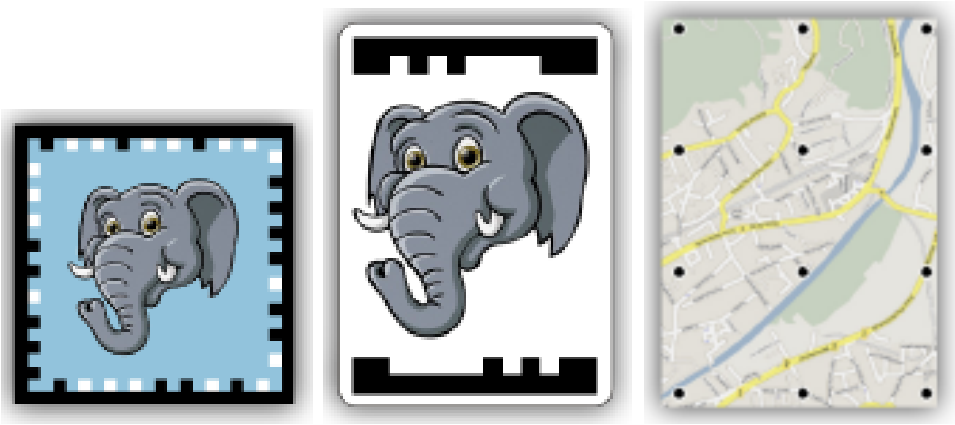
- *Frame markers.* 1.6(a) La robustezza del marker è ottenuta grazie al forte contrasto dei bordi neri che circondano l'immagine. I bordi hanno un pattern simile ad un barcode che semplifica e facilita il riconoscimento e la diversificazione del marker. Il contenuto del marker è variabile e viene inteso come una sorta di decorazione del marker, ma non influisce

sul riconoscimento del marker che è basato esclusivamente sul pattern che descrive il bordo.

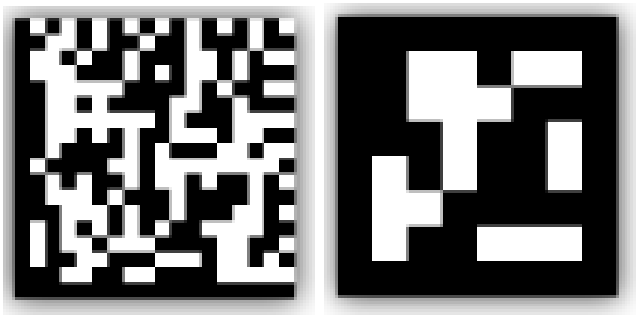
- *Split markers.* 1.6(b)Se il Frame Marker è caratterizzato da bordi che racchiudono l'immagine, questa tipologia di marker è caratterizzata da due barcode(superiore e inferiore), invece che quattro, riducendo l'area occupata dal marker, che può contenere anch'esso un' immagine al suo interno ininfluente sul processo di rilevamento.
- *Dot markers.* 1.6(c)Hanno il pregio di essere poco invasivi a livello di occupazione e impatto grafico sulla scena. Consistono in una griglia bidimensionale di punti circolari neri con un contorno bianco.
- *DataMatrix markers* (ISO/IEC16022). 1.6(d)DataMatrix è uno standard ISO per barcode 2D. Il barcode viene naturalmente utilizzato come marker e la sua facile customizzazione ne permette una forte distinzione, mentre i colori bianco e nero col quale viene rappresentato aiutano l'individuazione.
- *ID markers* (simple-id and BCH).1.6(e) Molto simile al Data matrix. Codifica un numero a 9-bit in un pattern 6*6. I 9 bits del numero sono ripetuti 4 volte per coprire i 36 bits che rappresentano il marker(6*6).

1.4.2 Tracking Markerless

L'Augmented Reality (AR) sviluppata negli ultimi dieci anni in vari settori dell'intrattenimento si è basata soprattutto sul riconoscimento di marker fiduciali come quelli bitonali appena descritti. Le ricerche recenti puntano ad ottenere un tracking *markerless* ovvero privo di marker 1.7), che renda l'interazione naturale e meno invasiva. Le tecniche più comuni si basano sull'estrazione di punti caratteristici dai frame che compongono la sequenza video ([15]). Le difficoltà maggiori si incontrano a fronte di fenomeni di oc-



(a) Frame Markers *(b)* Split Markers *(c)* Dot Markers



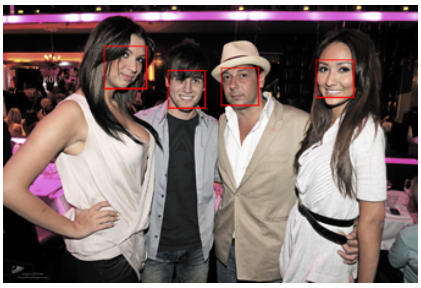
(d) DataMatrix Markers *(e)* SimpleID Markers

Fig. 1.6: Classificazione dei Markers

clusione, cambi di prospettiva, variazione di illuminazione, ecc... Alla base dell'implementazione di un sitema di tracking interattivo c'è il riconoscimento di uno o più oggetti all'interno della sequenza di frame. Per ogni oggetto vengono individuati, estratti e descritti alcuni punti di interesse. Lo scopo di questa fase è fornire una base di dati che caratterizzi l'oggetto e che dia una rappresentazione robusta rispetto alle problematiche di cui sopra. Analizzate le immagini di riferimento, si passa al confronto, frame per frame, con le immagini provenienti da una telecamera per individuare e localizzare gli oggetti ricercati. Il tutto deve essere svolto in poco tempo, tipicamente si richiede che la sequenza di operazioni venga effettuata più volte al secondo

(a) Tracking di oggetti(scatola di biscotti1)

(b) Tracking di oggetti(scatola di biscotti2)



(c) Riconoscimento facciale

Fig. 1.7: Alcuni esempi di markerless tracking

per rendere l'esperienza dell'utente verosimile e non fastidiosa.

1.4.3 Tracking ibrido

Il tracking che viene descritto e implementato in questa tesi è un *tracking ibrido* che unisce gli aspetti positivi di quello basato sui marker binari e di quello basato su features naturali. La differenza tra questo tracciamento ed uno puramente markerless sta nel fatto che in questo viene limitata l'area da analizzare all'interno di un pattern quadrato, come per i marker fiduciali, sgravando il sistema dall'analisi dell'intero frame. L'intero frame viene comunque scandito per poter estrarre i contorni neri che racchiudono i marker, ma è un processo meno oneroso rispetto a quello risultante dall'applicazio-

ne di algoritmi di estrazione e descrizione di punti caratteristici sull'intero frame. Individuati i pattern quadrati con bordo nero vengono analizzate le immagini contenute. Le immagini fungono da identificatore, come avviene nei SimpleID Markers 1.6(e), con la differenza che, al posto di un ID a 9 bits, all'interno della regione quadrata vengono processati oggetti con caratteristiche naturali. Il marker naturale è di norma meno invasivo di quello fiduciale e allo stesso tempo il tracking risulta più veloce di quello markerless in quanto non si applicano algoritmi sull'intero frame ma solo sulle regioni di interesse individuate.

Capitolo 2

Algoritmi per Image Matching off-line

Il presente capitolo offre una panoramica dello stato dell'arte degli algoritmi per l'individuazione e descizione di *Local Points* (in letteratura anche definiti come *Feature Points o Characteristic points*). Particolare attenzione viene qui posta sugli algoritmi che per primi sono stati presentati in letteratura e che sono tipicamente caratterizzati da buone capacità descrittive ma anche da un elevato costo computazionale e che quindi vengono classificati come algoritmi off-line: *SIFT, GLOH, DAISY*.

2.1 Local Detectors e Local Descriptors nell' Image Matching

L'Image Matching è la computazione effettuata su un insieme di immagini che rappresentano lo stesso scenario, pur avendo caratteristiche differenti, per individuarne alcuni elementi invarianti (come in figura 2.1). La diversità tra i vari elementi di questo insieme, può essere data da:

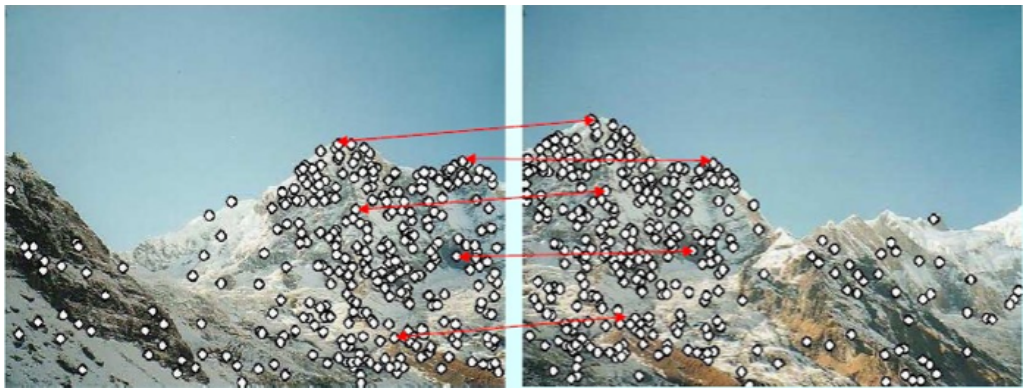


Fig. 2.1: Corrispondenze tra punti caratteristici individuati su immagini che rappresentano la stessa scena inquadrata da punti di vista differenti.

- un diverso punto di vista rispetto al quale l'immagine è stata acquisita,
- una differente rotazione,
- una scalatura,
- condizioni di illuminazione non uguali,
- differenti livelli di blurring applicati al set di immagini,
- distorsioni, effetti prospettici, scalature anisotropiche ecc.

Acquisito un set di immagini differenti, per tutte o alcune delle caratteristiche sopracitate, rappresentanti il medesimo oggetto, si estraggono alcuni punti caratteristici. Ogni punto(Interesting Points o Locals Points) sarà caratterizzato da una descrizione basata su vari criteri(tipicamente misure sui gradienti nell'intorno locale), che permetterà la distinzione rispetto agli altri punti della stessa immagine e l'identificazione del suo eventuale corrispondente all'interno delle altre immagini dell'insieme. In figura 2.2 si nota come non tutti i punti individuati trovano una corrispondenza, inoltre quelli che la trovano non sempre lo fanno in maniera corretta. Esistono diversi approcci per effettuare questa computazione, ognuno dei quali provvede a fornire una descrizione e individuazione in base ad alcune caratteristiche globali (velocità di computazione, accuratezza, specificità per alcuni task).

Fig. 2.2: Ricerca dei punti caratteristici su un murales fotografato da due diversi punti di vista (nei cerchi gialli i punti individuati con un corrispettivo nell'altra immagine, in blu quelli senza corrispondenza)

2.2 Panoramica e caratteristiche comuni dei vari algoritmi

In questa tesi vengono trattati sei tra i più importanti algoritmi di estrazione e descrizione di punti caratteristici:

- **SIFT** (Scale Invariant Feature Transform) è uno tra i primi algoritmi di individuazione e descrizione di Interesting Points descritti in letteratura. L'approccio è tra i più robusti, ma non è consigliabile per computazioni real-time.
- **GLOH** (Gradient Location Orientation Histogram) estende SIFT fornendo un' implementazione più accurata dei descrittori.
- **SURF** (Speed Up Robust Features) uno tra i più moderni algoritmi, caratterizzato da una buona robustezza e una discreta velocità di calcolo.
- **DAISY** utilizzato per il matching denso tra immagini, trova molte applicazioni nell' ambito della ricostruzione di ambienti 3D utilizzando una sequenza di immagini in ingresso e nel calcolo delle *depth map*.

- **BRIEF** utilizza FAST per l'estrazione di keypoints e BRIEF per la loro descrizione. La sua caratteristica principale è la velocità computazionale, ma non è altrettanto efficace nella robustezza non fornendo invarianza rispetto alla rotazione e alla scalatura.
- **ORB** estende BRIEF fornendo un'implementazione invariante rispetto alla rotazione e alla scalatura, mantenendo prestazioni simili in termini di velocità di BRIEF.

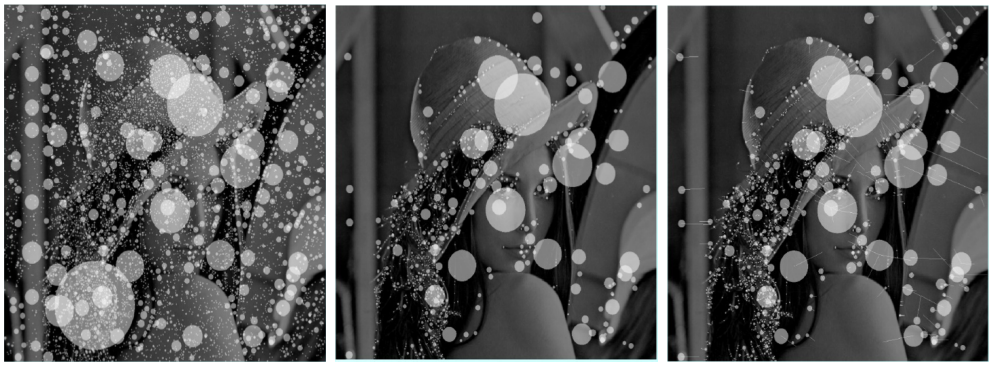
Possiamo distinguere in ognuno di questi algoritmi alcuni aspetti comuni:

- Individuazione dello SCALE SPACE: per ogni immagine dell'insieme viene creato uno Scale Space rappresentativo della scena da molti punti di vista, che fornisce la possibilità di ottenere invarianza sulla individuazione dei punti sulla stessa immagine ripresa da distanze diverse 2.3(a).
- Localizzazione degli Interesting points nello Scale Space: Calcolo per ogni livello dello scale space dei locals points caratteristici 2.3(b).
- Filtering basso contrasto ed edge: Filtraggio degli interesting points, calcolati al passo precedente, per scartare i punti individuati con un valore di contrasto, rispetto all'intorno dell'immagine, inferiore ad una soglia 2.3(c).
- Calcolo dei Local Descriptors: Calcolo e assegnamento di una direzione o vettore che sintetizzi il valore dei gradienti nell'intorno del punto 2.3(d).

2.3 SIFT (Scale Invariant Feature Trasform)

In questo paragrafo viene analizzato e studiato SIFT descritto in [12] e [13].

(a) Creazione Scale Space



(b) Localizzazione dei punti caratteristici

(c) Flitraggio dei punti

(d) Calcolo dei descrittori

Fig. 2.3: Passi principali degli algoritmi di Features Detection.

2.3.1 Scale Space Extrema Detection

Il primo passo dell'algoritmo consiste nella costruzione di uno Scale Space tramite un filtro Gaussiano applicato ricorsivamente ad ogni livello(partendo dall'immagine di iniziale), per generare il successivo. 2.4 Viene scelto un filtro Gaussiano, perchè fornisce invarianza, sia sulla scalatura, che sulla rotazione. Viene calcolato ogni livello con la seguente formula:

$$L(x, y; \sigma) = G(x, y; \sigma) * I(x, y)$$

dove $L(x, y; \sigma)$ è il valore in locazione (x, y), a livello di scala σ, calcolato come convoluzione([18]) tra $G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} * e^{\frac{-(x^2+y^2)}{2\sigma^2}}$, che è la funzione di Gauss in (x, y), con valore di scala σ e $I(x, y)$, che è il valore dell'immagine in (x, y). Costruito lo Scale Space dell'immagine in input, viene calcolata la *DoG (Difference of Gaussian)*, ovvero la differenza di due livelli di scala adiacenti, separati da un fattore moltiplicativo di scala k, come segue.

$$D(x, y; \sigma) = L(x, y; k\sigma) - L(x, y; \sigma)$$

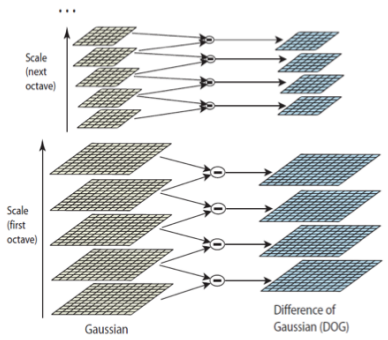


Fig. 2.4: Creazione dello Scale Space frutto della differenza tra i livelli di scala nello Scale Space originale.(DoG)

Questa operazione permette di calcolare un nuovo scale space nel quale ogni livello è ottenuto mediante differenze di due livelli adiacenti nello scale space originale. DoG è un ottima approssimazione della *Scale Normalized Laplacian of Gaussian* $\sigma^2\nabla^2 G$, che come dimostra Lindeberg(1994) in [26], è utile ad ottenere una invarianza di scala. Mikolajczyk(2002) in [21] afferma che i massimi e i minimi di $\sigma^2\nabla^2 G$ producono feature più stabili. Dopo aver calcolato DoG si effettua una ricerca su di esso, per individuare i minimi e massimi locali, facendo un confronto di ogni punto con i 26 suoi vicini (8 vicini sul suo livello, 9 sul livello superiore e 9 su quello inferiore). Incrementando il numero di campionamenti sulla scala, aumenta anche il numero di Local Points individuati e di conseguenza anche il numero di matching, ma aumenta

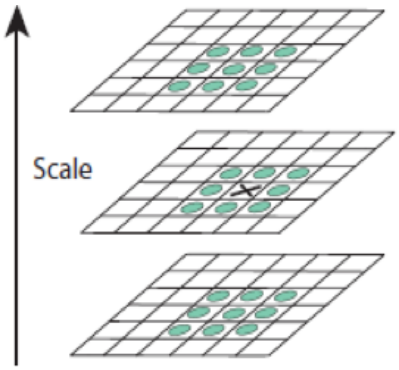


Fig. 2.5: Ricerca degli estremi sullo Scale Space: ogni punto viene confrontato con i 26 adiacenti.

anche il costo computazionale della ricerca. Per evitare costi eccessivi, nella ricerca dei Local Extrema, si cerca un buon compromesso tra efficienza e completezza. Risultati empirici dimostrano che, partendo da una scala di campionamento $\sigma = 1.6$, si può ottenere un buon risultato in termini di efficienza e di completezza. Inoltre, l'applicazione di un filtro di *Blur* sulla immagine di partenza, permette di eliminare alte frequenze di rilevamento degli estremi locali (come mostrato in figura 2.6).

Fig. 2.6: Utilizzo di filtri di Blurring per la riduzione della frequenza di rilevamento degli estremi locali. 2.6(a) Immagine non filtrata avrà un'alta frequenza di rilevamento di estremi locali. 2.6(b) Immagine filtrata avrà una frequenza di rilevamento minore

2.3.2 Filtraggio Interestings Points

Tra i punti candidati a diventare Local Points, vengono scartati quelli a **basso contrasto** rispetto all'intorno locale e quelli posizionati sugli **edge**, in quanto non altamente discriminanti e distintivi.

Filtraggio sul contrasto Viene calcolato lo sviluppo di Taylor della funzione di scale space $D(x, y, \sigma)$ facendo in modo che l'origine sia spostata sul punto campione.

$$D(s) = D + \frac{\partial D^T}{\partial s}s + \frac{1}{2}s^T\frac{\partial^2 D}{\partial s^2}s \text{ [1]}$$

D e le sue derivate sono valutate sul punto campione e $s = (x, y, \sigma)$ è l'offset da questo punto. La posizione dell'estremo è determinata prendendo la derivata di questa funzione su s

$$\hat{s} = -\frac{\partial^2 D^{-1}}{\partial s^2} * \frac{\partial D}{\partial s} \text{ [2]}$$

sostituendo [2] in [1] si ottiene

$$D(\hat{s}) = D + \frac{1}{2}\frac{\partial D^T}{\partial s}\hat{s} \text{ [3]}$$

La funzione [3] è usata per scartare gli estremi a basso livello di contrasto.

Filtraggio sugli edge Per la stabilità, non è sufficiente scartare i punti chiave con valore di contrasto basso, la DoG ha una forte risposta lungo gli *edge*, ovvero lungo i bordi. Un punto su un edge è poco distinguibile e quindi poco stabile, anche se lungo la direzione perpendicolare al bordo stesso ha alti valori di contrasto rispetto al suo intorno. Un picco così mal definito in DoG avrà una curvatura principale larga lungo il bordo e una piccola lungo la direzione perpendicolare. La curvatura principale può essere calcolata con una matrice Hessiana 2x2 H calcolata sul punto e sulla scala del punto caratterisitco rilevato:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Gli autovalori di H sono proporzionali alla curvatura principale di D. Presi a e b, rispettivamente, l'autovalore più grande e il più piccolo si ha

$$Tr(H) = D_{xx} + D_{yy} = a + b$$

$$Det(H) = D_{xx} * D_{yy} - (D_{xy})^2 = a * b$$

Dato il rapporto di due autovalori, r=a/b e quindi a=r*b, per eliminare dai candidati, i punti sugli edge, bisogna imporre che $\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$

2.3.3 Assegnamento dell'orientazione

Assegnando un'orientazione coerente a ciascun punto caratteristico, in base alle proprietà locali dell'immagine, esso viene descritto in maniera invariante rispetto alla rotazione. Il livello di scala al quale viene individuato il punto è lo stesso dal quale viene prelevata l'immagine L. Per ogni campione individuato $L(x, y)$, in posizione (x, y) , al dato livello di scala si definiscono $m(x, y)$ e $\theta(x, y)$ rispettivamente come il magnitudo del gradiente e la sua orientazione. Questi ultimi calcolati come segue:

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y)))$$

Con questi dati si costruisce un istogramma delle orientazioni con 36 ingressi corrispondenti a 36 punti campione presi attorno al punto chiave, che discretizzano il range di 360 gradi di orientazioni possibili del gradiente. Ogni ingresso dell'istogramma è pesato col valore di magnitudo del gradiente e con una finestra circolare Gaussiana con un σ pari a 1.5 volte il valore di scala al

quale è stato individuato il punto caratteristico. Picchi nell'istogramma delle orientazioni corrispondono a direzioni dominanti dei gradienti locali. Per creare un' orientazione stabile, da fornire al punto, vengono utilizzati i picchi di magnitudo più alto e tutti quelli maggiori di un valore corrispondente al 80% del picco maggiore. In locazioni con picchi multipli, di grandezza simile, ci saranno punti chiave multipli nella stessa posizione e scala, ma con orientazione differente.

2.3.4 Descrittore locale dell'immagine

Le operazioni descritte in precedenza forniscono invarianza su scala, posizioni e orientazioni. Il passo successivo è quello di fornire una rappresentazione del punto che garantisca invarianza su altri tipi di trasformazioni come cambi di illuminazione o cambi di punti di vista o prospettiva. A questo scopo viene utilizzata una rappresentazione basata sul modello di visualizzazione biologica, in particolare sui neuroni della corteccia visuale primaria, che permettono di riconoscere e confrontare oggetti in uno spazio tridimensionale da differenti punti di vista. L'immagine 2.7 illustra il calcolo del descrittore del

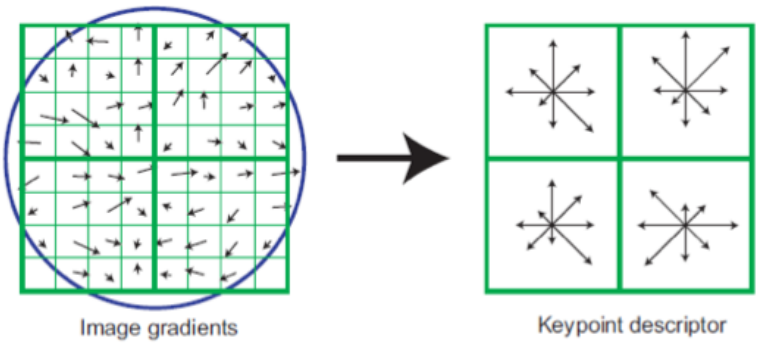


Fig. 2.7: Somma dei gradienti e sintesi in istogrammi rappresentativi dei Descrittori dei punti caratteristici.

punto chiave. Viene calcolato m e θ, come indicato nella sezione precedente,

attorno alla posizione del punto, usando la scala dello stesso per selezionare il livello di Blur Gaussiano per l'immagine. I gradienti dell'immagine a sinistra sono accorpati in sottoregioni 4*4 e sintetizzati in istogrammi. La lunghezza di ogni freccia corrisponde alla somma dei magnitudo dei gradienti che hanno una direzione simile all'interno della stessa sottoregione (come mostrato nell'immagine a destra). Allo scopo di ottenere invarianza sulla rotazione le coordinate del descrittore e le orientazioni dei gradienti sono ruotate relativamente all'orientazione del punto. Per efficienza computazionale vengono pre-calcolati i gradienti () per tutti i livelli della piramide.

Prima di assegnare un peso al magnitudo di ogni sample point si utilizza una funzione di *smussamento* Gaussiano (illustrata nell'immagine con una finestra circolare nella parte sinistra), con σ una volta e mezza la grandezza della finestra del descrittore. Con il passo precedente si cerca di evitare cambi improvvisi nel descrittore, dovuti a piccole variazioni nella posizione della finestra, dando allo stesso tempo meno importanza a gradienti lontani dal centro del descrittore. Un campione del gradiente, come mostrato nella sinistra dell'immagine, può essere shiftato di 4 posizioni e continuare a contribuire allo stesso istogramma, come mostrato nella parte destra, questa caratteristica dell'algoritmo è fondamentale per mantenere l'invarianza sugli spostamenti. Inoltre viene applicata un'interpolazione trilineare per distribuire il valore di ogni gradiente negli ingressi degli istogrammi adiacenti. Il descrittore è formato da un vettore contenente i valori di tutti gli ingressi dell'istogramma delle orientazioni. Nella figura viene mostrato un array 2*2 di istogrammi, ma risultati empirici dimostrano che i migliori risultati si ottengono utilizzando array 4*4 all'interno dei quali vengono memorizzati istogrammi con 8 orientazioni ognuno. Per ogni punto caratteristico viene costruita una struttura dati array contenente 4*4*8=128 elementi. Infine il

vettore viene processato per ottenere invarianza sui cambi di illuminazione:

- in un primo passo il descrittore viene normalizzato, questa fase permette di eliminare le variazioni dovute ai cambiamenti di contrasto (le variazioni di contrasto consistono in moltiplicazioni di ogni pixel per valori costanti).
- l' invarianza sui cambi di luminosità, in cui ad ogni pixel viene aggiunto un valore costante, è ottenuta automaticamente grazie al fatto che i valori dei gradienti sono ottenuti tramite differenze di pixel.
- grossi mutamenti nel magnitudo $m(x, y)$ dei gradienti provocati dai cambiamenti di illuminazione non lineari come la saturazione della camera oppure da differenti orientazioni di superfici tridimensionali vengono ridotti applicando una soglia di grandezza e ri-normalizzando il vettore.

2.4 GLOH (Gradient Local Orientaton Histogram)

GLOH, descritto in [27] e [21], è un approccio che estende SIFT, incrementandone la robustezza e la precisione nella localizzazione dei punti caratteristici. GLOH cambia il modo di rappresentare la zona di campionamento circostante il punto caratteristico. SIFT costruisce attorno al Key Point un istogramma a 3 dimensioni 4*4*8, dove la regione 4*4 individua la posizione della regione nella quale avviene il campionamento, mentre gli 8 valori in ingresso in ognuna delle 16 celle della griglia descrivono l'orientazione del gradiente. GLOH, a differenza di SIFT, calcola il descrittore in una regione rappresentata da una griglia circolare con tre ingressi in direzione radiale e 8 ingressi in direzione angolare. Questa tecnica utilizza 17 ingressi (8+8+1),

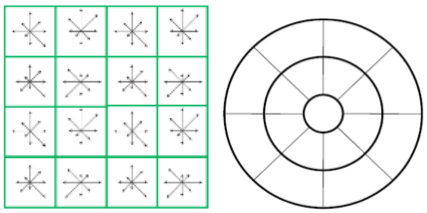


Fig. 2.8: Rappresentazione dei descrittori SIFT e GLOH

ognuno dei quali conterrà un vettore di 16 orientazioni, ottenendo così un istogramma formato da 272 ingressi.

2.4.1 Valutazione delle prestazioni

I criteri di valutazione sui quali si basa il giudizio su un algoritmo rispetto ad un altro sono più di uno. Gli algoritmi presentati in questa tesi sono classificati in base ad alcune caratteristiche principali:

- Velocità di calcolo: la caratteristica fondamentale di un sistema che garantisce lo svolgimento di compiti in tempo reale è la velocità della computazione, che lo rende interattivo e responsivo.
- Precisione e potenza descrittiva: un sistema che si occupa di ricostruire ambienti tridimensionali utilizzando campioni bidimensionali, dello stesso, ripresi da angolazioni differenti, spesso non ha la pretesa di dover svolgere tale compito in un tempo breve, quindi in questo caso si predilige la potenza descrittiva e si sacrifica la velocità della computazione.
- Tipologia di invarianza garantita: alcuni algoritmi, sviluppati per determinate tipologie di task, non garantiscono tutti i tipi di invarianza(rotazione, scalatura, cambi di prospettiva..). Questa caratteritica rende tali approcci poco utilizzabili in determinati ambiti, ma specializzati e più performanti in altri dove tali requisiti non sono richiesti.

Le migliorie apportate da GLOH, rispetto a SIFT, sono da individuare nella migliore robustezza e precisione nella descrizione. Dare una valutazione

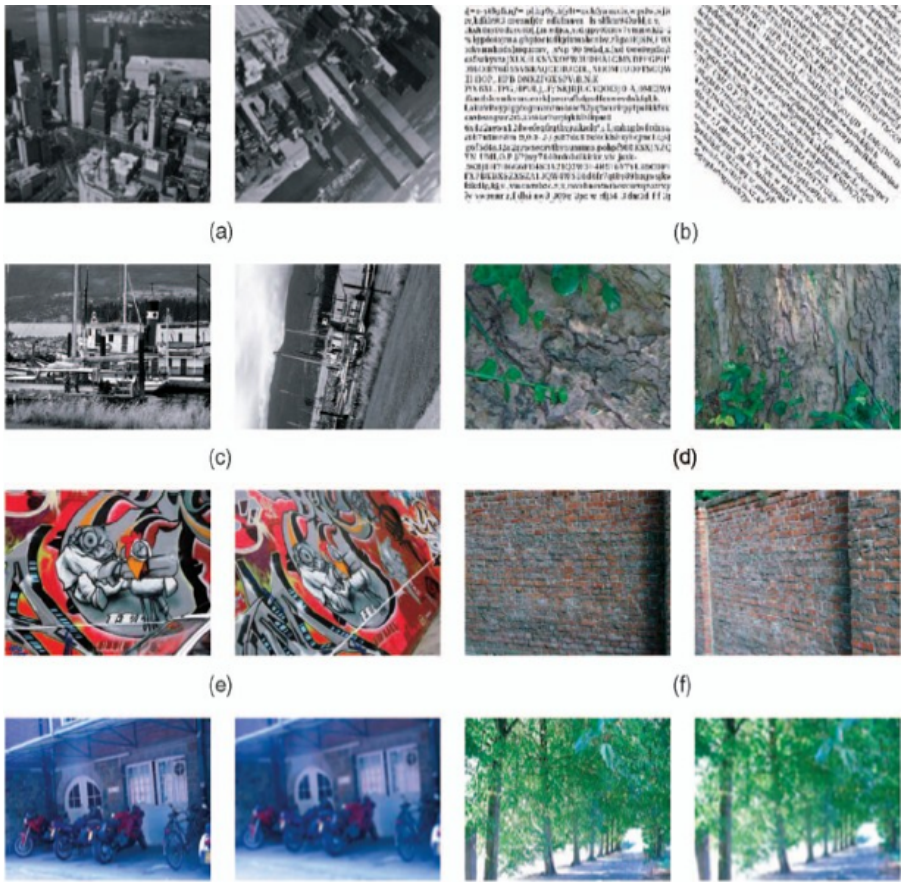


Fig. 2.9: Insieme delle coppie di immagini utilizzate nei test.

dei due algoritmi in termini di numero di corrispondenze corrette, vuol dire prendere un set di coppie di immagini(come raffigurato nell'immagine), in cui ogni coppia di immagini rappresenta la stessa scena, e le immagini che formano la coppia stessa differiscono per alcune trasformazioni applicate alla scena originale (rotazione, illuminazione, scalatura...). Nelle coppie di sinistra sono raffigurate scene con caratteristiche strutturate, mentre in quelle di destra sono presenti scene caratterizzate da motivi ripetuti. Per la comparazione ci si basa sul numero di confronti corretti e di falsi positivi ottenuti per ogni

coppia di immagini. Due regioni A e B coincidono se la distanza tra i due descrittori D_A e D_B è minore di un certo threshold *d*. A tal proposito si definiscono i valori *recall* e *1-precision*. Il primo è il numero di corrispondenze corrette rispetto il numero di regioni corrispondenti tra due immagini.

$$recall = \frac{correctMatches}{correspondences}$$

Il numero di confronti corretti e corrispondenze è determinato per mezzo dell' *overlap error*, ovvero un indice di corrispondenza tra due regioni che hanno subito una trasformazione. Overlap error è definito come il rapporto tra l'intersezione e l'unione delle due regioni A e B.

$$\epsilon_S = 1 - \frac{(A \cap H^T B H)}{(A \cup H^T B H)}$$

H è l'omografia [1] delle due immagini. Il confronto è corretto se $\epsilon_S < 0.5$. $1 - precision$ rappresenta il numero di falsi positivi rispetto ai confronti totali.

$$1 - precision = \frac{falseMatches}{correctMatches + falseMatches}$$

Dati i valori di $recall$, $1 - precision$ e $corrispondences$, il numero di confronti corretti è dato da $correspondences * recall$ e il numero di falsi positivi è $correspondences * recall * (1 - precision)/precision$

Dimensionalità La dimensionalità del descrittore influisce molto sul potere distintivo dello stesso e sulla sua robustezza. GLOH è calcolato su 17 posizioni e per ognuna vengono calcolate 16 orientazioni utilizzando i 128 autovettori più grandi(128 è il valore preferibile empiricamente dimostrato),

[1] In matematica e geometria una omografia è una relazione tra punti di due spazi tali per cui ogni punto di uno spazio corrisponde ad uno ed un solo punto del secondo spazio. Il problema dell'omografia bidimensionale consiste nella determinazione di una trasformazione in grado di mappare punti di un piano in punti di un altro piano.[2]

mentre PCA-SIFT che è la variante ottimizzata di SIFT ne usa 36. Nella tabella in figura 2.10 viene calcolata la somma dei primi 10 autovalori per ogni descrittore trattato. Questo valore è altamente indicativo del potere distintivo del descrittore. Con i criteri appena descritti si valutano le presta-

TABLE 1
Distinctiveness of the Descriptors

Descriptor	$\sum_{i=1:10} eigenvalue(i)$	$\sum_i eigenvalue(i)$
PCA-SIFT	1.0839e+12	1.9743e+12
GLOH	1.4085e+11	2.8277e+11
SIFT	3.4210e+09	6.4541e+09
Shape context	3.3582e+09	7.1149e+09
Spin images	4.4791e+09	5.2355e+09
Cross correlation	1.0657e+09	1.4076e+09
Steerable filters	4.1529e+07	4.2909e+07
Differential invariants	2.5970e+07	2.6349e+07
Complex filters	1.6328e+07	1.8264e+07
Moments	1.3829e+07	1.8100e+07

Fig. 2.10: Distintività dei descrittori. Somma dei primi 10 e di tutti gli altri autovalori per differenti descrittori

zioni di SIFT e GLOH in base alle caratteristiche delle scene.
Le scene che contengono motivi similari necessitano di un descrittore con un elevato potere distintivo. in questo caso GLOH è preferibile rispetto a SIFT.Bisogna precisare che questa valutazione può variare a seconda della trasformazione applicata(la trattazione accennata riguarda le trasformazioni affini e non quelle derivate da cambiamenti di illuminazione, dall'applicazione di un filtro di blur...). Va precisato inoltre che influisce sulle prestazioni dei vari approcci anche l'algoritmo di detecting(Harris affine detector, Hessian affine detector...) dell'interesting points. Per una più completa trattazione rimandiamo alla lettura dell'articolo A Performance Evaluation of Local Descriptors[22].

2.5 DAISY An Efficient Dense Descriptor Applied to Wide-Baseline Stereo

DAISY, ampiamente descritto in [27] e [28], è differente dagli algoritmi studiati precedentemente, non solo per quel che riguarda la sua implementazione, ma soprattutto per il campo di utilizzo e lo scopo per il quale è stato creato. DAISY è un approccio utilizzato nel dense wide-baseline matching, ovvero nel matching per pixel(denso) di immagini che rappresentano la medesima scena, ma in condizioni di vista estremamente differenti. Questo algoritmo

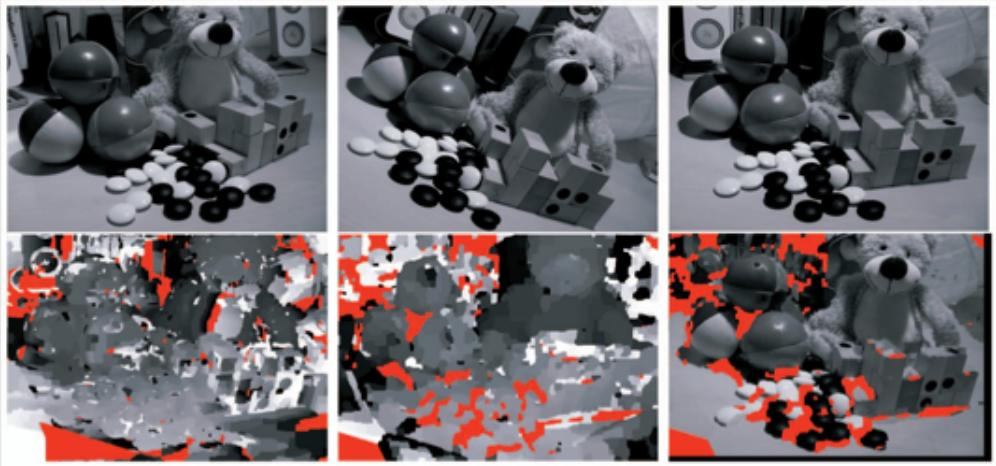


Fig. 2.11: Costruzione di mappe di profondità con DAISY

trova largo impiego nella creazione di mappe di profondità per la ricostruzione di ambienti tridimensionali. Lo *shot-baseline stereo matching*, ovvero il confronto tra aree di immagini che non discostano molto l'una dall'altra, è un problema ampiamente trattato, mentre la sua controparte *wide-baseline* è ancora oggetto di studi, in quanto più complessa. La maggiore complessità è dovuta alle grosse distorsioni prospettiche e all'incremento delle aree occluse, che spesso portano un pixel in un' immagine a non avere il corrispettivo nell'altra (problema poco rilevante nello shot-baseline).

Prima di DAISY, nel *wide-baseline stereo matching*, sono state utilizzate molte tecniche, come le *correlation windows*, che consistono nell'analisi di porzioni(all'interno delle finestre) delle due immagini e nel confrontare le frequenze delle variazioni di intensità di pixel su ogni riga tra una finestra e l'altra, per determinare se le porzioni di immagine rappresentano la stessa parte di scena. Questo approccio ha lo svantaggio che, usando grandi correlation windows, si perde in robustezza rischiando di accavallare aree di profondità diverse, mentre usando correlation window piccole servono immagini di ottima qualità per rendere affidabile il processo, senza considerare la sensibilità ai cambi di illuminazione e ai pattern ripetitivi. DAISY scarta a priori le correlation window, facendo posto ai local descriptor simili a quelli discussi fino ad ora. SIFT e GLOH sono robusti grazie al calcolo di istogrammi delle orientazioni dei gradienti, ma sono ideati per il matching sparso, quindi sono computazionalmente onerosi per quello denso, SURF(descritto nel capitolo successivo), pur essendo veloce da calcolare per pixel non tiene conto dello schema di pesi spaziale, dando uguale peso ad ogni gradiente nel calcolo del descrittore (nel calcolo denso questo si tramuta in grossi artefatti).

2.5.1 DAISY Descriptor

Delle tre dimensioni degli istogrammi costruiti nei descrittori per SIFT e GLOH, due identificano la posizione spaziale nell'immagine del punto e la terza identifica la direzione del gradiente nel punto stesso, inoltre ogni pixel appartenente alla regione locale contribuisce, in base alla sua distanza dal punto di interesse e in base al suo gradiente, alla costruzione del descrittore. Ogni ingresso dell'istogramma è incrementato del valore della direzione del gradiente moltiplicata per un peso inversamente proporzionale alla distanza tra il pixel e il punto di interesse. Di conseguenza l'istogramma conterrà una

somma pesata delle direzioni dei gradienti attorno al centro. Nel descrittore DAISY, le convoluzioni dell'immagine originale con derivate orientate di filtri Gaussiani, prendono il posto delle somme pesate. Una *convolved orientation map* si calcola con la seguente formula:

$$G_O^{\sum} = G_{\sum} \cdot \left(\frac{\partial I}{\partial o}\right)^+$$

dove $G_{\sum}$ è un kernel Gaussiano, o è l'orientamento della derivata e l'operatore $(.)^+$ indica che se il valore all'interno alle parentesi è negativo viene settato a 0. Si costruisce il descittore leggendo i valori nelle *convolved orientation map*. Si definisce la orientation map come $G_o = \left(\frac{\partial I}{\partial o}\right)^+$. Il calcolo delle convolved orientation maps di dimensioni differenti viene fatto a basso costo in quanto convoluzioni con kernel grandi possono essere ottenute da convoluzioni successive con kernel più piccoli. In particolare avendo calcolato $G_o^{\sum_1}$ si ottiene $G_o^{\sum_2}$ con $\sum_2 > \sum_1$ come segue:

$$G_O^{\sum_2} = G_{\sum_2} \cdot G_o = G_{\sum} \cdot G_{\sum_1} \cdot G_o = G_{\sum} \cdot G_O^{\sum_1}$$

con $\sum = sqrt\sum_2^2 - \sum_1^2$

In sintesi il calcolo del descrittore DAISY consiste nei seguenti passi: Sull'immagine in input, vengono calcolate 8 orientation map G, una per ogni direzione da rappresentare, dove $G_o(u, v)$ è il gradiente dell'immagine in posizione (u, v) e direzione o se più grande di zero, altrimenti uguale a zero. Di ogni orientation map viene effettuata una convoluzione riepetuta con kernel gaussiani di dimensioni differenti per ottenere convolved orientation maps per regioni di diverse grandezze(il calcolo può essere svolto in maniera efficiente con convoluzioni ricorsive). L'immagine 2.12 descrive lo stadio dell'algoritmo nel quale per ogni pixel vengono scelte 8 direzioni lungo le quali calcolare orientation maps per diverse regioni. Ogni cerchio nell'immagine descrive una

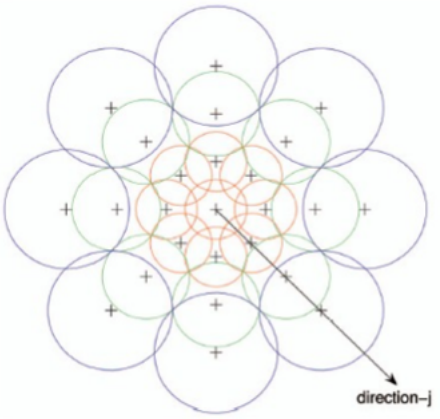


Fig. 2.12: Rappresentazione del descrittore DAISY

regione con raggio proporzionale alla deviazione standard del kernel Gaussiano e i '+' indicano le posizioni nelle quali le convolved orientation maps vengono campionate, il centro è il pixel dove viene calcolato il descrittore. Sovrapponendo queste regioni si ottiene uno smooth tra esse, questo garantisce transizioni uniformi tra una regione e l'altra. DAISY è un vettore contente valori che arrivano dalle convolved orientation maps.

$$h_{\sum}(u,v) = [G_1^{\sum}(u,v), ..., G_8^{\sum}(u,v)]$$

e il suo corrispettivo normalizzato è $\tilde{h}_{\sum}(u,v)$. Il descrittore DAISY completo $D(u_0,v_0)$ in posizione (u_0,v_0) è una concatenazione di vettori $\tilde{h}$

$$\begin{aligned} D(u_0,v_0) = [&\tilde{h}_{\sum_1}(u_0,v_0),\ \tilde{h}_{\sum_1}(l_1(u_0,v_0,R_1)), ..., \tilde{h}_{\sum_1}(l_N(u_0,v_0,R_1)),\\ &\tilde{h}_{\sum_2}(l_1(u_0,v_0,R_2)), ..., \tilde{h}_{\sum_2}(l_N(u_0,v_0,R_2)),\\ &\tilde{h}_{\sum_3}(l_1(u_0,v_0,R_3)), ..., \tilde{h}_{\sum_3}(l_N(u_0,v_0,R_3))]^T \end{aligned}$$

dove $l_j(u,v,R)$ è la posizione con distanza R da (u,v) in direzione $j \in [1,...,N], N = 8$.

Capitolo 3

Algoritmi per Image Matching interattivo

Il presente capitolo offre un'analisi dello stato dell'arte di algoritmi per l'estrazione e la descrizione di Local Points focalizzandosi però sugli approcci caratterizzati da una velocità di calcolo superiore rispetto a quelli presentati nel capitolo 2. Questi approcci, generalmente più recenti in letteratura, sono maggiormente adatti ad un'impiego per applicazioni real-time pur offrendo in alcuni scenari applicativi, una potenza descrittiva comparabile o superiore.

3.1 SURF (Speed Up Robust Features)

SURF(trattato in [10]) è uno tra i più moderni e utilizzati algoritmi per detection e description di punti caratteristici. Fornisce *features* invarianti alla rotazioni e alle scalature. Questi due tipi di invarianza sono un buon compromesso tra complessità e robustezza, assumendo che l'aggiunta di un secondo grado di complessità che comprenda invarianze riguardanti scalature anisotropiche e distorsioni prospettiche spesso da un guadagno visibile solo

su grandi cambiamenti di punti di vista, apportando un impatto negativo sui tempi di calcolo. In alcuni ambiti, per guadagnare in velocità, si trascura l'invarianza rotazionale ottenendo così un descrittore U-SURF(upright SURF). Il largo successo e utilizzo di SURF è motivato dall' incremento di performance sui tempi di calcolo, ottenuto senza penalizzare troppo la robustezza e la precisione dello stesso. Il miglioramento è attribuibile all'utilizzo di immagini integrali per effettuare le operazioni di convoluzione. L'impiego di questa tecnica riduce il costo dovuto alla creazione dello scale space con DoG o convoluzioni di La Place. L'algoritmo si può sintetizzare in due punti fondamentali:

- *individuazione di interest points* in differenti e distintive locazioni dell'immagine (corner, blobs, T-junctions). Questi punti devono essere ripetibili sotto differenti condizioni di vista.
- *rappresentazione dell'intorno* di ogni interest points tramite un descrittore che sarà distintivo e robusto al rumore.

3.1.1 Immagini Integrali

La velocità raggiunta in Surf è dovuta all'utilizzo di Immagini Integrali (3.1 descritte in [19]), ovvero rappresentazioni intermedie delle immagini originali. Le immagini integrali sono calcolate, rapidamente, da un'immagine in input e usate per accelerare il calcolo delle somme dei gradienti di una qualsiasi area rettangolare con vertice superiore sinistro nell'origine. Data una qualsiasi immagine I in input al punto di coordinate (x,y), l'immagine integrale corrispondente S è calcolata tramite la somma dei valori dell'intensità dei pixel compresi tra il punto e l'origine (posta nell'angolo superiore sinistro):

$$S(x, y) = \sum_{i=0}^{i<=x} \sum_{j=0}^{j<=y} I(x, y)$$

Usando questa rappresentazione il compito di calcolare la somma dei gradienti all'interno di una regione rettangolare è ridotto a 4 operazioni. La

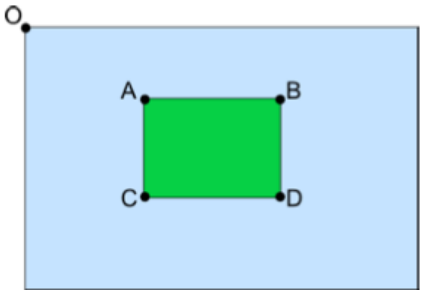


Fig. 3.1: Calcolo della somma dei gradienti all'interno di una porzione di immagine usando un'immagine integrale

somma delle intensità dei pixel nel rettangolo ABCD è calcolata come:

$$\sum = A + D - (C + B)$$

Dove A, B, C e D sono i valori integrali nei vertici del rettangolo. Il tempo di calcolo è invariante al cambio di grandezza dell'area. SURF fa uso di questa proprietà per calcolare velocemente la convoluzione al variare della dimensione dei filtri applicati all'immagine.

3.1.2 Fast Hessian

Il detector SURF è basato sul determinante della matrice Hessiana. Per motivare questa scelta bisogna descrivere le proprietà di questo tipo di matrici. Si consideri una funzione continua in due variabili $f(x, y)$. H (*Hessian matrix*) è la matrice delle derivate parziali di f.

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial x \partial y} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{bmatrix}$$

Il suo determinate(discriminante) è calcolato come

$$det(H) = \frac{\partial^2 f(x,y)}{\partial x^2} \frac{\partial^2 f(x,y)}{\partial y^2} - \left(\frac{\partial^2 f(x,y)}{\partial x \partial y} \right)^2$$

ed è usato per classificare massimi e minimi estremi locali della funzione. In particolare se il determinante è:

- NEGATIVO, allora gli autovalori hanno segno opposto, quindi non ci sono estremi locali.
- POSITIVO, gli autovalori sono entrambi positivi o entrambi negativi, allora il punto è un estremo locale.

Per riportare la teoria delle matrici Hessiane all'ambito dell'elaborazione delle immagini si sostituisce $f(x, y)$ con $I(x, y)$ che è l'intensità del pixel in posizione (x, y). Il calcolo delle derivate parziali del secondo ordine dell'immagine è effettuato mediante convoluzione con un appropriato nucleo. Nel caso di SURF il filtro utilizzato è quello Gaussiano del secondo ordine normalizzato, che permette un'analisi oltre che nello spazio, anche su diverse scale. Si costruiscono i nuclei del filtro in x, y e xy e si calcolano quindi i quattro termini della matrice Hessiana. Utilizzando la Gaussiane si varia l'effetto di smooth durante la fase di convoluzione in modo che il determinante sia calcolato a differenti livelli. Essendo inoltre, la gaussiana, una funzione isotropica(quindi a simmetria circolare), la convoluzione col suo nucleo è invariante alla rotazione. Si calcola la matrice Hessiana come funzione di spazio $p = (x, y)$ e scala σ. Dato un punto $p = (x, y)$, in un immagine I, la matrice $H = (p, \sigma)$, in un punto p, a livello di scala σ, è definita come

$$H(p, \sigma) = \begin{bmatrix} L_{xx}(p, \sigma) & L_{xy}(p, \sigma) \\ L_{xy}(p, \sigma) & L_{yy}(p, \sigma) \end{bmatrix}$$

dove $L_{xx}(p, \sigma)$ è la convoluzione tra derivata Gaussiana del secondo ordine $\frac{\partial^2}{\partial x^2} g(\sigma)$ e l'immagine I, nel punto p. Queste derivate sono conosciute come Laplaciane di Gauss. Si calcola il determinante di H per ogni pixel per verificare che sia un estremo o meno. Per ottenere un incremento di prestazioni si considera un'approssimazione di Laplacian of Gaussian otte-

nuta con i box filters calcolati velocemente grazie alle immagini integrali. Per quantificare la differenza, in termini di prestazioni, possiamo considerare il numero di accessi nell'array e le operazioni richieste nella convoluzione. Per un filtro 9*9 sono necessari 81 accessi nel filtro originale, mentre solo 8 operazioni per la rappresentazione semplificata. Anche l'incremento della grandezza del filtro non influisce sulla velocità di calcolo nella versione approssimata, mentre nell'approccio originale rende più oneroso il calcolo. Nei

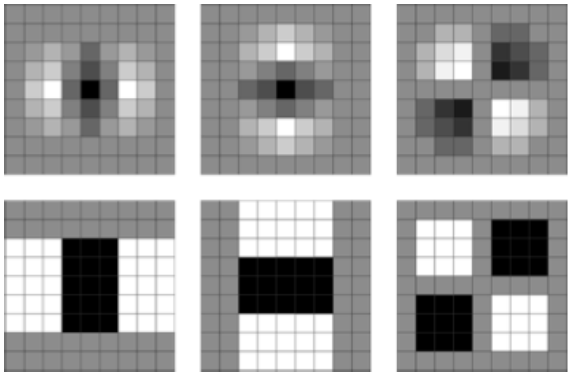


Fig. 3.2: Rappresentazione di filtri ottenuti come derivate seconde di filtri gaussiani nella parte superiore dell'immagine e discretizzazione tramite box filters nella parte inferiore.

box filters illustrati nell' immagine 3.2 ci sono in alto i filtri 9*9 ottenuti da derivate seconde di kernel Gaussiani con $\sigma = 1.2$ in direzione $x, yexy$ e in basso la loro approssimazione. Si indicano le approssimazioni come D_{xx}, D_{yy}, D_{xy}. I pesi applicati alle regioni rettangolari vengono semplificati per ragioni di efficienza computazionale, inoltre vengono bilanciati i relativi pesi nell'espressione per il calcolo del determinante della matrice Hessiana con $\frac{|L_x y(1.2)|_F |D_x x(9)|_F}{|L_x x(1.2)|_F |D_x y(9)|_F} = 0.912 \approx 0.9$ dove $|x|_F$ è la norma di Frobenius. Questo produce $det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2$. Lo Scale Space solitamente viene implementato come una piramide di immagini ripetutamente processate con un filtro Gaussiano e in seguito sottocampionate per ottenere un nuovo livello della piramide. Grazie all' utilizzo delle immagini integrali si può evita-

re di applicare iterativamente lo stesso filtro all'output del livello precedente, applicando invece filtri di ogni dimensione alla stessa velocità all'immagine originale, parallelizzando il calcolo. Sintetizzando lo scalespace viene calcolato modificando(sovra-scalando) le dimensioni del filtro, invece che riducendo la grandezza dell'immagine come illustrato nell'immagine seguente, dove a sinistra viene rappresentato lo scale space tradizionale, mentre a destra lo scale space implementato nell'approccio SURF 3.3. Anche in SURF lo scal-

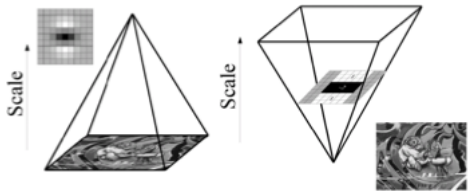


Fig. 3.3: SIFT Scale Space e SURF Scale Space

space è diviso in un numero di ottave, dove ogni ottava fa riferimento ad una serie di risposte a filtri, fino al raddoppiamento della loro scala. Partendo dall'output del filtro 9*9 come livello di scala iniziale, corrispondente alla derivata Gaussiana con $\sigma = 1.2$, i successivi livelli vengono calcolati filtrando l'immagine con una maschera gradualmente più grande (9*9, 15*15, 21*21, 27*27). Aumentando la dimensione dei filtri, aumenta anche la scala di ri-

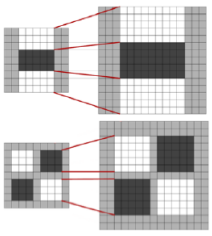


Fig. 3.4: Box Filters verticale D_{yy} e verticale/orizzontale D_{xy} per differenti scale 9*9 e 15*15.

ferimento e poichè si mantengono le proporzioni la si può calcolare con la seguente formula.

$$\sigma_{approx} = CurrentFilterSize * \frac{BaseFilterScale}{BaseFilterSize} = CurrentFilterSize * \frac{1.2}{9}$$

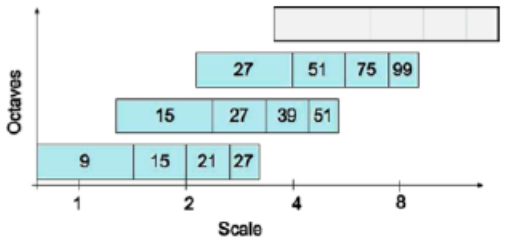


Fig. 3.5: Prime tre ottave dello scale space

Come mostrato in figura 3.5, da un' ottava alla successiva l'incremento dei filtri raddoppia passando da 6 a 12 a 24(...), quindi, mentre nella prima ottava i filtri nelle varie scale saranno 9*9, 15*15, 21*21(...), nella seconda ottava le scale saranno 15*15, 27*27, 39*39(...) e nella terza saranno 27*27, 51*51, 75*75,(...)

3.1.3 Localizzazione degli interest point

Come avviene per SIFT e GLOH le risposte vengono filtrate rispetto a valori di soglia di contrasto sotto i quali i punti vengono scartati. Successivamente viene scelto un set di punti candidati,facendo il solito confronto tra i punti e i 26 vicini come descritto in figura 2.5.

3.1.4 SURF Descriptor

Il descrittore Surf definisce come le intensità dei gradienti dei pixel sono distribuite nell' intorno del punto di interesse individuato dalla Fast Hessian. Questo approccio è simile a quello SIFT, ma l'utilizzo delle immagini integrali congiunto a quello dei filtri Haar-wavelet garantisce una maggiore velocità e robustezza della rappresentazione. I filtri Haar-wavelet sono utilizzati per trovare gradienti nelle direzioni x e y. Nell' immagine 3.6 il filtro di sinistra calcola la risposta orizzontale e il destro quella verticale. I pesi sono rispetti-

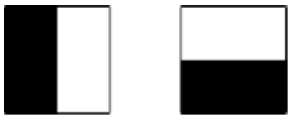


Fig. 3.6: Filtri Haar-Wavelet in direzione orizzontale e verticale

vamente -1 per la regione nera e 1 per quella bianca. Il calcolo del descrittore è così definito:

- Fissare un orientamento basato sulle informazioni residenti in un intorno circolare del punto di interesse.
- Costruire una regione quadrata, allineata con l'orientamento precedentemente assegnato, dalla quale estrarre il descrittore.

Assegnamento dell'orientazione Per ottenere invarianza rotazionale ad ogni punto di interesse individuato viene assegnata un' orientazione ripetibile sotto varie condizioni di vista. Per determinarla si calcola la risposta della Haar-wavelet grande 4σ su un'insieme di pixel contenuti in un intorno circolare di raggio 6σ (dove σ è la scala nella quale il punto è stato individuato). Calcolate le risposte vanno pesate con un filtro Gaussiano ($\sigma = 2.5 * s$) centrato sul punto di interesse. Una volta pesate vengono rappresentate come punti in uno spazio vettoriale con la risposta orizzontale lungo l'ascissa e quella verticale lungo l'ordinata. L'orientamento dominante viene stimato calcolando le somme delle risposte all'interno di una finestra a orientazione variabile di un angolo $\pi/3$. Le risposte verticali e orizzontali nella finestra vengono sommate per formare un nuovo vettore. Il vettore più lungo da la sua orientazione al punto di interesse come mostrato nella figura 3.7

Descrittore Per estrarre il descrittore si seleziona una regione quadrata attorno al punto di interesse(figure 3.8 e 3.9), con lato 20 volte il livello di scala al quale è stato selezionato il punto e orientamento uguale a quello calcolato

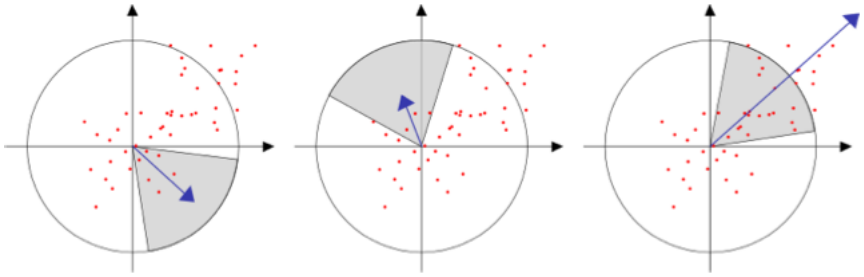


Fig. 3.7: Rappresentazione nello spazio vettoriale delle risposte Haar-wavelet dei pixel nell'intorno circolare del punto caratteristico.Il vettore più grande viene utilizzato per l'assegnazione di un orientamento.

al passo precedente. La regione viene suddivisa in 4*4 sotto-regioni quadrate

Fig. 3.8: Descrittori SURF rappresentati come finestre orientate nell'immagine murales.

e per ogni sotto-regione ottenuta si prendono, a intervalli regolari spaziati in una griglia 5*5, dei punti campione nei quali calcolare risposta orizzontale e verticale (d_x d_y),in base alla specifica orientazione della finestra. All'interno di ognuna delle 16 sotto-regioni si sommano le 5*5 = 25 risposte per le due direzioni e i loro valori assoluti, ottenendo un vettore a 4 dimensioni per ogni sotto-regione, quindi un descrittore v del punto di interesse di dimensione 4*16 = 64 come descrittodi seguito:

$$v = (\sum d_x, \sum |d_x|, \sum d_y, \sum |d_y|)$$

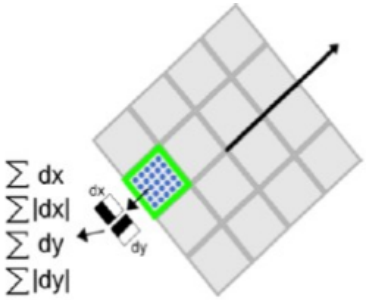


Fig. 3.9: Struttura del descrittore SURF.

Per dare un'idea più chiara di come vengono calcolate le risposte Haar-wavelet si riporta un esempio grafico in figura 3.10. In una regione omogenea, come

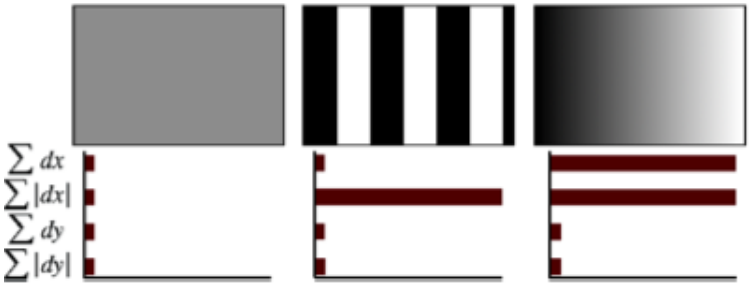


Fig. 3.10: Tre differenti risposte Haar-Wavelet su immagini con andamenti sulle intensità dei gradienti diversi(la prima ha un andamento omogeneo, la seconda alternato in direzione orizzontale, la terza incrementale in direzione orizzontale).

quella rappresentata nella parte sinistra dell'immagine 3.10, le risposte nelle due direzioni sono basse in quanto i valori dei gradienti sono stabili, nella regione centrale i cambiamenti di intensità si alternano e quindi si annullano nella risposta orizzontale, ma sono alti nei valori assoluti, mentre nella risposta verticale l'intensità è costante. Nell'ultima regione si vede un incremento costante di intensità, di conseguenza i due valori della risposta orizzontale sono alti.

3.2 BRIEF (Binary Robust Independent Elementary Features)

BRIEF è il nome dell'approccio utilizzato per fornire una descrizione ai local points individuati. Il primo passo effettuato dall'algoritmo è quello di individuare i punti caratteristici rappresentati dai cosidetti *corners* 3.11, ovvero punti nei quali il gradiente presenta discontinuità in più direzioni e che per loro natura si prestano bene ad essere utilizzati come *feature points*. Esistono vari algoritmi adatti a questo scopo (Moravec, Harris, FAST, ...)

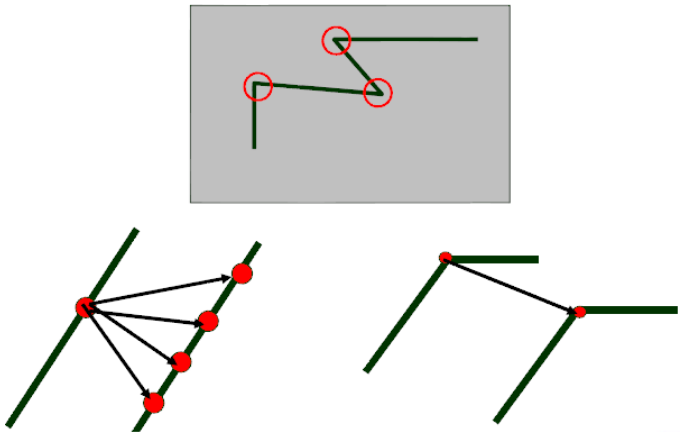


Fig. 3.11: Punti di discontinuità in due direzioni, definiti come corner, vengono utilizzati come punti caratteristici in BRIEF.

alcuni dei quali analizzano i cambi di direzione rapidi degli *edge*, all'interno dell'imagine, calcolando i massimi locali sulle curve viste come funzioni, altri esaminano piccoli pezzi di immagine(*patches*), cercando quelli simili a corners. In BRIEF viene utilizzato un detector di tipo FAST (descritto in [24]) che appartiene alla famiglia degli algoritmi che analizzano le *patches*.

3.2.1 FAST (Features from Accelerated Segment Test) Detector

Si consideri un cerchio di sedici pixel attorno al corner candidato p 3.12. Il detector classifica p come corner se esiste un insieme di n pixel contigui sulla circonferenza che sono più chiari dell'intensità I_p più una soglia t del pixel candidato o più scuri di $I_p - t$. La scelta di $n = 12$ permette di effettuare un

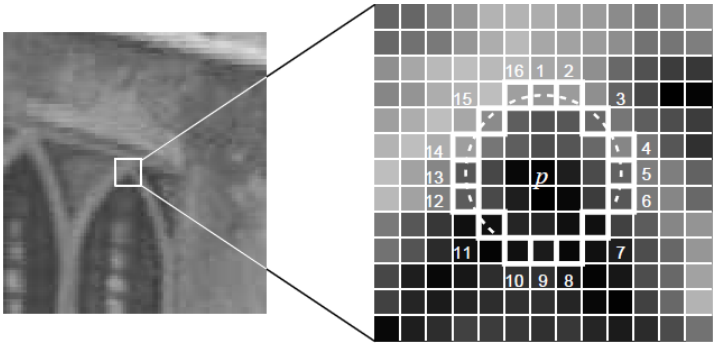


Fig. 3.12: Individuazione del corner con un test veloce sui 16 punti che circondano il candidato.

test veloce che esclude un gran numero di corners falsi: il test esamina solo i 4 pixel 1, 5, 9 e 13 e se p è un corner allora almeno tre di questi deve essere più chiaro di $I_p + t$ o più scuro $I_p - t$, se ciò non accade viene scartato e il test completo su tutta la circonferenza viene fatto solo su quelli che passano questo test preliminare.

3.2.2 BRIEF Descriptor

Il descrittore BRIEF([11]) consiste in una stringa di bit rappresentativa di ognuna delle patches contenenti i corners individuati al passo precedente. Il confronto tra le stringhe di bit viene effettuato anch'esso in maniera efficiente tramite la Hamming distance per mezzo di una operazione *XOR* dei bit. Ogni bit della stringa viene valorizzato in base a un test tra intensità di pixel

all'interno della patch p di grandezza $S * S$

$$\tau(p; x, y) = \begin{cases} 1, & \text{se } p(x) < p(y) \\ 0, & \text{altrimenti} \end{cases}$$

, dove p(x) è l'intensità del pixel nella versione smooth di p al punto $x = (u, v)^T$. Si seleziona un set di n_d coppie di locazioni (x, y) 3.13 da testare e si forma una stringa di bit a n_d dimensioni che rappresenterà il descrittore seguendo la seguente formula:

$$f_{n_d}(p) = \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(p; x_i, y_i)$$

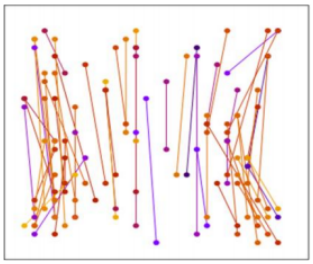


Fig. 3.13: Rappresentazione del descrittore BRIEF definito come insieme di test binari tra punti attorno al corner individuato.

Considerando $n_d = 128$, 256 e 512 si generano stringhe lunghe n_d, selezionando n_d coppie di locations (x_i, y_i) all'interno della patch, assumendo l'origine della stessa nel suo centro. Le coppie di locations vengono selezionate all'interno della porzione di dimensione S*S come segue:

$$(X, Y) \sim Gaussian(0, \tfrac{1}{25} S^2)$$

BRIEF non è progettato per essere invariante rispetto alla rotazione e alla scalatura, ma può essere modificato nella procedura di identificazione e in quella di descrizione per ovviare a queste mancanze, mantenendo comunque ottime prestazioni riguardo i tempi di calcolo e aumentando il suo potere distintivo, raggiungendo quello di approcci cosiddetti robusti come SIFT e SURF.

3.3 ORB (Oriented Fast and Rotation-Aware Brief)

ORB estende BRIEF fornendo un descrittore più robusto e comunque performante in termini di velocità come descritto in [25]. I principali contributi

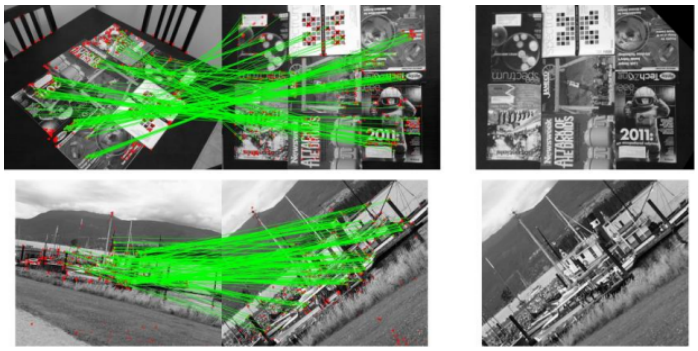


Fig. 3.14: Image Matching con ORB su un set di scene caratterizzato da variazioni di prospettiva e rotazioni.

che questo approccio fornisce sono:

- Aggiunta di una componente di orientamento al FAST detector.
- Calcolo efficiente di features BRIEF orientate.

3.3.1 oFAST (FAST Keypoint Orientation)

FAST non fornisce una misura qualitativa del corner (*cornerness*), per ovviare a questa mancanza ORB applica il metodo di Harris che fornisce un peso ai corner individuati, per poi ordinarli in base a tale misura scegliendo i migliori N, inoltre FAST non produce features multiscala, quindi per ottenere invarianza sulla scalatura viene applicata la stessa computazione a una piramide di immagini. L'invarianza rotazionale si ottiene calcolando l' *intensità del centroide*, come descritto in [23] e considerando l'intensità del corner come la differenza dall'intensità del suo centro. Il valore così definito

viene utilizzato per imputare al corner un orientamento. Viene definito il *momento* della patch come:

$$m_{pq}(p) = \sum_{x,y} x^p y^q I(x, y)$$

Il centroide è definito come:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

Con questi valori viene costruito il vettore dal centro del corner O al centroide ($\vec{OC}$) e definito l'orientamento della patch utilizzando $atan2$:[1]

$$\theta = atan2(m_{01}, m_{10})$$

Per incrementare l'invarianza sulla rotazione si impone che i momenti vengano calcolati utilizzando x e y all'interno di una regione circolare di raggio r. Empiricamente viene scelta r come la grandezza della patch quindi x e y vengono scelti in un range[-r, r].

3.3.2 rBRIEF Rotation-Aware Brief

Nell'implementazione vengono prese in considerazione stringhe di bit lunghe $n = 256$ (BRIEF-64). L'immagine viene pre-filtrata con un filtro di *smooth*. In questa implementazione lo smooth è ottenuto usando le immagini integrali, dove ogni punto da testare è una sottoregione 5x5 della patch grande 31x31 pixel.

[1] In trigonometria la funzione a due argomenti atan2 rappresenta una variazione dell'arcotangente. Comunque presi gli argomenti reali x e y non nulli, atan2(y,x) indica l'angolo in radianti tra l'asse positivo delle X in un piano e un punto di coordinate (x,y) giacente su di esso. L'angolo è positivo se antiorario (upper half-plane, y>0) e negativo se in verso orario (lower half-plane, y < 0). [3]

Steered Brief Il calcolo di un descrittore per un insieme di copie della stessa patch ruotate è troppo costoso. Un approccio più efficiente è quello di *orientare o sterzare* Brief secondo la direzione del keypoint (steered brief). Per ogni insieme di n features(test binari) nelle posizioni (x_i, y_i), viene definita una matrice 2 x n $S = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}$ Usando l'orientamento della patch θ e la corrispondente matrice di rotazione R_θ si ottiene la versione sterzata S_θ di S: $S_\theta = R_\theta S$ Questa variante di Brief diviene $g_n(p, \theta) = f_n(p)|(x_i, y_i) \in S_\theta$. Viene costruita una tabella di ricerca di descrittori Brief precalcolati incrementando l'angolo iniziale di $2\pi/30$(12 gradi). Finché l'orientamento θ del keypoint è coerente, il corretto insieme di punti S_θ viene utilizzato per calcolare il proprio descrittore.

Varianza e Correlation Una delle proprietà importanti di BRIEF è che ogni feature ha una varianza grande e una media vicino a 0.5. La figura 3.15 mostra l'andamento della media per un tipico BRIEF di 256 bit su oltre 100k keypoints. La media di 0.5 dà la varianza di campionamento massima di 0.25 per le features. Possiamo notare che una volta orientato(sterzato) BRIEF lungo la direzione del keypoint, le medie assumono una distribuzione più uniforme questo rende le features meno distintive 3.15. Una varianza alta nei test rende le features più discriminanti, dal momento che rispondono in maniera differente in base agli input. Un'altra proprietà importante di Brief è quella di avere test scorrelati, affinchè ognuno contribuisca in maniera consistente al risultato.

Learning Good Binary Features Per recuperare la perdita di varianza insita nell'approccio Steered Brief e ridurre la correlazione tra i test binari, viene sviluppato un metodo di apprendimento per la scelta di un buon sot-

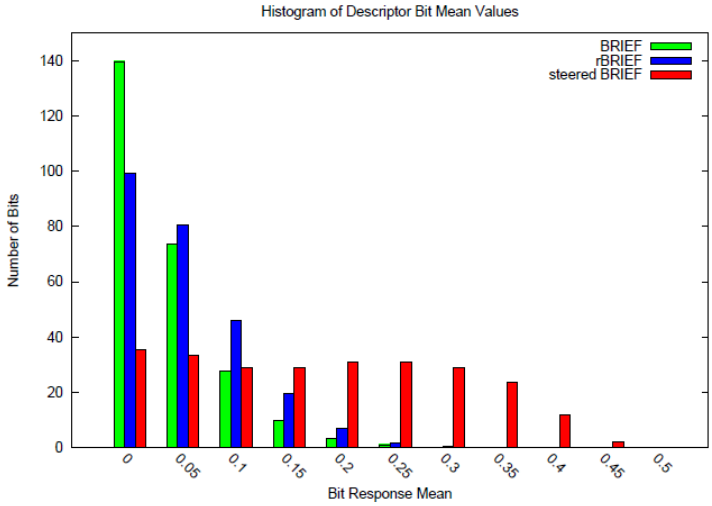


Fig. 3.15: Distribuzione delle medie dei descrittori BRIEF, rBRIEF e steered BRIEF, quest'ultimo ha un andamento uniforme delle medie rispetto ai primi due

toinsieme dei test stessi. Le nuove features sono composte da una grande numero di test binari. All'interno di questo insieme bisogna cercare, tra tutti i possibili test binari, quelli che hanno varianza elevata(e media vicina a 0.5) e un livello di correlazione basso. Il metodo è il seguente: Viene creato un insieme di 300k keypoints, si enumerano tutti i possibili test binari tratti da una patch di 31×31 pixel. Ogni test è formato da una coppia di sottofinestre di dimensione 5×5. In questo modo si ottengono $N = (31 - 5)^2$ possibili sottofinestre. Si vogliono selezionare coppie di due tra queste sottofinestre, quindi si hanno $\binom{N}{2}$ possibili test binari, dai quali si eliminano i test sovrapposti ottenendo $M = 205590$ test. Questo è l'algoritmo:

- Eseguire ogni test.
- Ordinare i test rispetto la distanza dalla media (0.5) formando il vettore T.
- Ricerca Greedy:[2]

[2]Un algoritmo greedy è un algoritmo che cerca di ottenere una soluzione ottima da un punto di vista globale attraverso la scelta della soluzione più golosa (aggressiva o avida, a seconda della traduzione preferita del termine greedy dall'inglese) ad ogni passo locale.

– Inserire il primo test nel vettore risultante R e rimuoverlo da T.
– Prendere il prossimo test da T e compararlo con tutti quelli in R. Se il suo valore di correlazione è maggiore di una soglia scartarlo altrimenti aggiungerlo a R.
– Ripetere il test precedente finchè non si ottengono 256 test in R. Se si ottengono meno di 256 valori, abbassare la soglia e riprovare.

Il risultato è chiamato rBRIEF e apporta un miglioramento significativo per quel che riguarda la varianza e la correlazione rispetto Steered Brief.

3.4 Conclusioni sullo studio dello stato dell'arte

La trattazione effettuata porta a fare alcune considerazioni necessarie al completamento dello studio. L'applicazione di questi algoritmi va valutata e determinata in base al contesto in cui si applicano e alla funzione che hanno all'interno di esso. Lo studio qui effettuato non determina un approccio migliore in assoluto rispetto agli altri. In base a quanto detto in precedenza, approcci come SIFT e GLOH sono molto robusti e distintivi nell' individuazione e nella descrizione dei punti di interesse, ma onerosi riguardo la velocità di calcolo. Essi possono essere collocati nell'ambito del riconoscimento offline di pattern caratteristici su immagini statiche ad alta definizione(anche se l'avanzare delle tecnologie e delle prestazioni delle nuove GPU e CPU potrà smentire a breve questa affermazione). SURF ha il pregio di saper coniugare precisione e potenza distintiva, con prestazioni in termini di velocità di

Questa tecnica consente, dove applicabile (infatti non sempre si arriva ad una soluzione ottima), di trovare soluzioni ottimali per determinati problemi in un tempo polinomiale [6]

calcolo migliori rispetto GLOH e DAISY, questo rende l'algoritmo applicabile nella realizzazione di strumenti per l'object detection, motion detection, pattern recognition in sequenze video. DAISY ha un campo di utilizzo differente, di fatti viene impiegato principalmente per la ricostruzione di scene tridimensionali, da sequenze di fotogrammi. BRIEF offre prestazioni migliori in termini di velocità, ma non garantendo l'invarianza sulla scalatura e sulla rotazione ha un campo di utilizzazione piuttosto ristretto. Infine ORB soddisfa le richieste di precisione e potenza descrittiva di cui molti task necessitano e mantiene una velocità di calcolo paragonabile a quella di BRIEF. Il grafico presentato di seguito fornisce una valutazione sull'invarianza rotazionale ottenuta da alcuni tra gli approcci descritti. La trattazione effettuata in [9] mette in evidenza le difficoltà presentate da BRIEF a fronte di trasformazioni di rotazione, come risulta anche dai test effettuati nel presente lavoro e descritti al capitolo 6, mentre ORB risulta essere il miglior compromesso tra velocità e precisione tra quelli descritti.

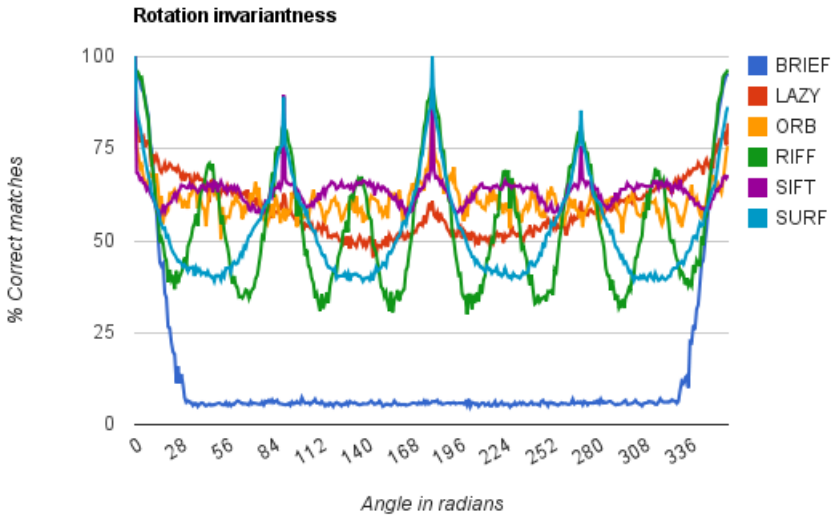


Fig. 3.16: Invarianza su Rotazione di alcuni algoritmi di features extraction descritti in questa tesi

Capitolo 4

Fiducial Marker Detection

In questo capitolo viene descritto un tipico procedimento di Fiducial Marker Detection facendo una breve panoramica su ArUco, la libreria utilizzata ed estesa per implementare il tracker descritto in questa tesi. Il procedimento descritto è fondamentale ai fini del presente lavoro, in quanto fornisce la struttura sulla quale si appoggia il sistema di tracciamento implementato. Un Fiducial non è altro che un'immagine bidimensionale, generalmente di forma quadrata o circolare, che possa essere identificata in maniera univoca. Il seguente schema rappresenta le varie fasi di un sistema di tracking basato su Fiducial bitonali 4.1:

4.1 Ricerca del Fiducial

La ricerca del Fiducial all'interno della scena visualizzata in ciascun frame è, ovviamente, la prima, fondamentale operazione che deve essere eseguita in una qualunque applicazione di Realtà Aumentata. Il Fiducial deve poter essere trovato all'interno dell'intero campo visivo, in qualunque posizione esso si trovi, quindi deve essere riconosciuto anche se appare in maniera distorta

(inquadrato con una certa inclinazione). Per questo motivo è desiderabile che le informazioni contenute nel Fiducial non siano troppo dense: questo fatto inficerebbe inoltre il riconoscimento di Fiducial posti ad una certa distanza dalla videocamera. Per quanto riguarda la forma di un Fiducial, è preferibile che questa sia rettangolare (o, meglio, quadrata), in quanto, come vedremo in seguito, per la procedura di pose-estimation sono necessari come minimo quattro punti distinti. Una volta che all'interno del frame viene trovato il Fiducial, occorre calcolare la posizione relativa della videocamera rispetto ad esso ed il suo orientamento. Questa procedura, definita pose estimation, richiede la conoscenza di alcuni parametri della videocamera, per ottenere i quali occorre eseguire una procedura di calibrazione prima di poter procedere all'acquisizione vera e propria dei frame. La conoscenza di tali parametri è importante perchè ciascuna videocamera possiede caratteristiche diverse, inoltre non sempre acquisisce un'immagine nella maniera desiderabile, ma tale immagine è soggetta a una certa distorsione provocata dalle lenti e dal modo in cui l'immagine reale a tre dimensioni viene proiettata su un piano bidimensionale.

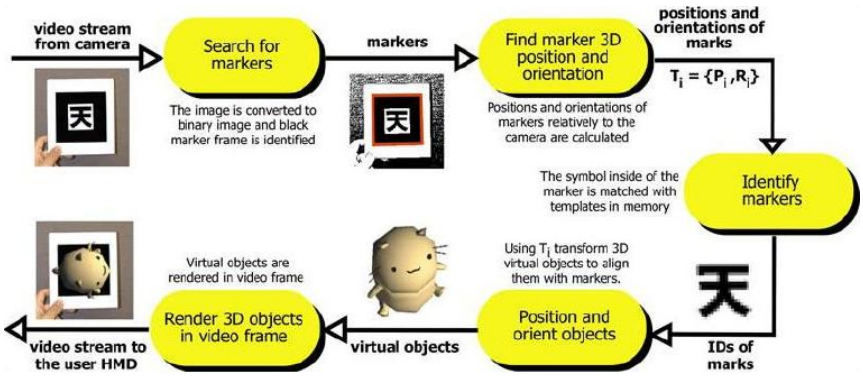


Fig. 4.1: Fasi del Fiducial tracking.

4.2 Procedura di calibrazione

La calibrazione permette di trovare sia la matrice dei parametri intrinseci che il vettore di distorsione della videocamera, dati necessari per la pose estimation. Esistono diverse librerie che propongono diversi metodi di calibrazione(tutti comunque seguono le stesse linee generali), come per esempio la routine cvCalibrateCamera2() di OpenCV, che è la libreria utilizzata nel presente lavoro di tesi. La calibrazione viene eseguita inquadrando con la videocamera un'immagine bidimensionale raffigurante una struttura regolare (tipicamente una scacchiera oppure una griglia) da diverse angolazioni.Oltre ad ottenere i valori intrinseci e di distorsione della videocamera, vengono anche ricavati la posizione e l'orientamento dell'oggetto di calibrazione: cioè la matrice dei parametri estrinseci composta da un vettore di traslazione e da una matrice di rotazione, questa matrice è la stessa che permette di visualizzare le immagini virtuali nella posizione e orientamento corretti in base al Fiducial inquadrato. In pratica, durante la calibrazione viene effettivamente fatta anche una pose estimation della videocamera rispetto alla figura usata per calibrarla. Il motivo per cui viene utilizzata una scacchiera come oggetto di calibrazione è che, essendo costituita da quadrati bianchi e neri di dimensione nota, è difficile confondere un punto della scacchiera con un altro, anche riprendendola da angolazioni diverse, inoltre è più agevole identificare la posizione degli angoli necessari a risolvere il sistema di equazioni che permette di trovare tutti i parametri che occorrono alla calibrazione della videocamera. In generale qualunque immagine di un oggetto ripresa da una videocamera è in grado di fornire la posizione dell'oggetto nei confronti della videocamera in termini di rotazione e traslazione. Per ulteriori dettagli riguardanti la matrice di rotazione, il vettore di traslazione e la composizione della matrice di trasformazione (matrice dei parametri estrinseci), si rimanda

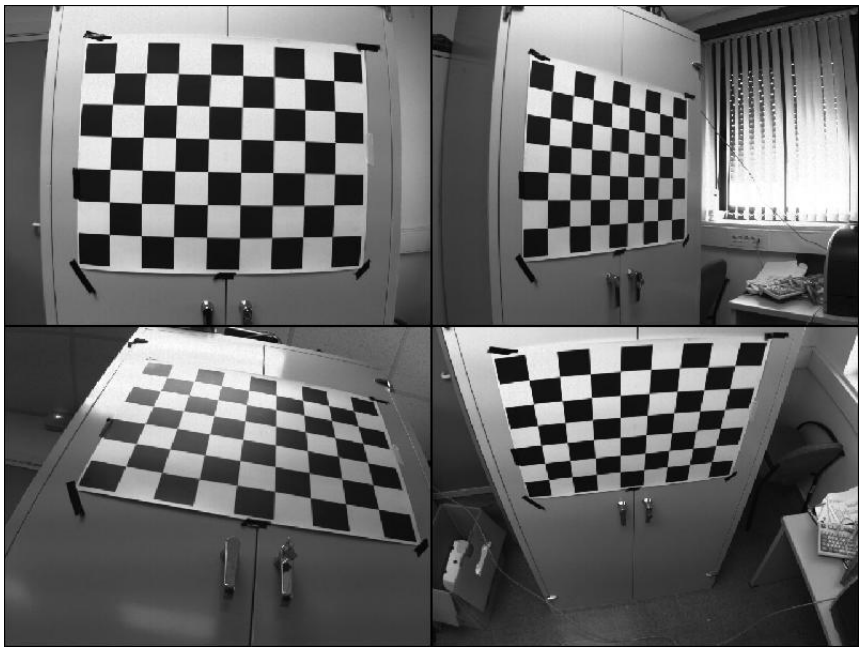


Fig. 4.2: Scacchiera utilizzata per la calibrazione.

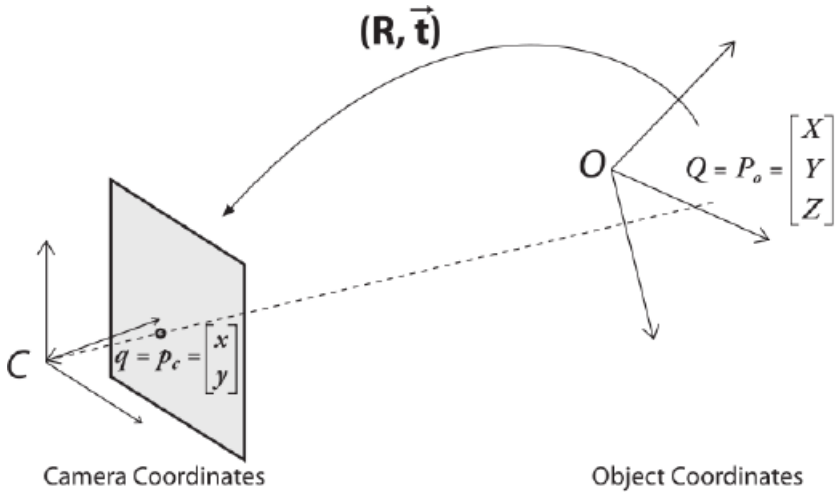


Fig. 4.3: Passaggio dal sistema di coordinate dell'oggetto al sistema di coordinate della videocamera.

all'appendice(A) o alla trattazione in [16].

4.3 Pose Estimation

La procedura di pose estimation (descritta in [17]) ha come obiettivo quello di proiettare i punti fisici di un oggetto (nel caso del tracking, l'oggetto in questione è il fiducial) sul piano immagine della videocamera, in modo da trovare le coordinate corrette sulle quali visualizzare un'immagine tridimensionale, la quale, in base all'orientamento della videocamera in relazione all'oggetto, deve anche apparire con l'orientamento corretto. Per il funzionamento di una qualunque procedura di pose estimation è necessario conoscere:

- i punti fisici (image points) da proiettare: i quattro vertici del fiducial.
- i punti del modello tridimensionale corrispondenti ai punti fisici (object points).
- i vettori di rotazione e di traslazione (matrice dei parametri estrinseci) per conoscere la posizione relativa della camera nei confronti del fiducial.
- la matrice contenente i parametri intrinseci della videocamera e i coefficienti di distorsione, ottenuti mediante la procedura di calibrazione

Quindi una volta noti i dati ottenuti tramite la calibrazione è possibile proiettare i quattro vertici del fiducial sul piano immagine: occorre soltanto ricavare i vettori di rotazione e traslazione del fiducial in modo tale da poter calcolare la matrice di trasformazione prospettica, nello stesso modo in cui è stata calcolata relativamente alla scacchiera di calibrazione. Esistono diversi algoritmi di questo tipo già implementati in diverse librerie grafiche. Prendendo ancora una volta la libreria OpenCV come riferimento, si accennerà di seguito al funzionamento di un algoritmo di pose estimation implementato in essa: POSIT.

POSIT è un acronimo che sta per Pose from Orthography and Scaling with Iteration: si tratta di un algoritmo iterativo in grado di stimare la posizione

e l'orientamento di oggetti tridimensionali di dimensioni note. La prima fase dell'algoritmo, detta POS, partendo dalla matrice dei parametri intrinseci calcola la matrice di trasformazione prospettica (in maniera uguale a come fatto per la calibrazione) relativa all'oggetto, la quale fornisce una prima stima di posizione ed orientamento. La seconda fase (IT) consiste semplicemente nell'iterare la procedura, fino a soddisfare un criterio di arresto definito al momento della chiamata della routine. Tipicamente l'algoritmo converge dopo quattro o cinque iterazioni. In molti casi, al posto di POSIT è sufficiente utilizzare la stessa funzione utilizzata per ottenere i parametri estrinseci dalla scacchiera di calibrazione (utilizzata tra l'altro all'interno dell'implementazione OpenCV di POSIT), senza dover necessariamente iterare il procedimento: tale funzione è cvFindExtrinsicCameraParams2. Passando come argomento a questa funzione le matrici contenenti gli object points, gli image points e i dati di calibrazione (parametri intrinseci e di distorsione) vengono calcolati il vettore di traslazione e la matrice di rotazione rispetto al fiducial inquadrato. In entrambi i casi, per ottenere questi risultati, devono essere trovati sulla superficie dell'oggetto almeno quattro punti non-complanari. Di questi punti viene ignorata la profondità, ovvero si suppone che abbiano le stesse coordinate sull'asse z, e si tiene conto di un fattore di scala s calcolato in base alla distanza dalla videocamera.

4.4 Visualizzazione del modello 3D.

Una volta trovata la matrice di trasformazione prospettica tramite la procedura di pose estimation, occorre capire quale tra i modelli tridimensionali presenti nel database del programma di Realtà Aumentata vada visualizzato in corrispondenza del fiducial. Occorre quindi poter riconoscere i fiducial in

maniera univoca, per esempio utilizzando dei codici ottici binari, in modo da poter associare a ciascun identificativo del fiducial un unico modello tridimensionale. La visualizzazione di un oggetto tridimensionale su un supporto bidimensionale quale è uno schermo avviene attraverso un processo definito *pipeline grafica* consistente in più fasi che trasforma la scena tridimensionale in un frame a due dimensioni. La prima fase, detta *Trasformazione* riceve co-

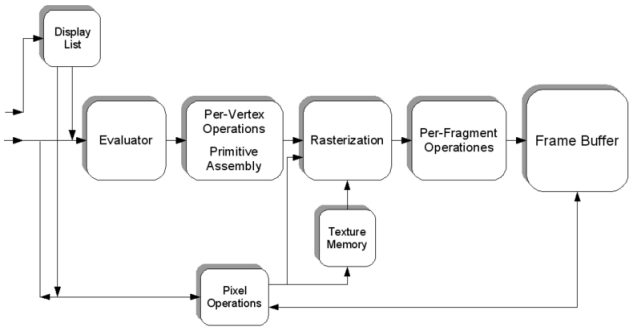


Fig. 4.4: Fasi della pipeline grafica in OpenGL.

me input i dati riguardanti i vertici, i lati e le facce dei poligoni dell'oggetto da visualizzare. In questa fase viene utilizzata la matrice di trasformazione prospettica che contiene i parametri di scala, rotazione e traslazione che permettono la visualizzazione dell'oggetto con il corretto orientamento. Nella seconda fase (*per-vertex lighting*) viene gestita l'illuminazione della scena 3D: vengono considerate le sorgenti di luce presenti nella scena e le proprietà riflettenti del modello tridimensionale per poter visualizzare il modello illuminato in maniera coerente. La fase di *viewing transformation* modifica il sistema di coordinate in modo da attenersi alla posizione e all'orientamento del punto di vista (la videocamera). La procedura di pose estimation, come già visto, fornisce i dati necessari a tale passaggio di coordinate. Successivamente vengono generate nuove primitive grafiche (fase di *primitives generation*) che sostituiscono quelle proprie del modello ricevuto come input

dalla pipeline e vengono mappate sul piano nella posizione conforme al punto di vista della videocamera (*projection transformation*). Durante questa fase viene considerata anche la distanza dell'oggetto dall'osservatore in modo che, in presenza di più oggetti sulla scena, si possa capire quale tra questi venga coperto dall'altro (*clipping*) ed evitarne così la visualizzazione. L'ultima fase (*rasterizzazione*) consiste nel generare i pixel dell'immagine bidimensionale in maniera conforme ai dati del modello 3D, oppure possono essere utilizzate delle texture per colorare i pixel dell'immagine raster. Se un'applicazione vuole accedere alla pipeline grafica, occorre gestire l'hardware necessario alla sua implementazione. A questo scopo esistono diverse librerie, tra le quali le più note sono *OpenGL* e *Direct3D*, che presentano il vantaggio di essere indipendenti dall'hardware.

4.5 Detection Process in Aruco

ArUco è una libreria opensource, sviluppata dal gruppo di ricerca *Aplicaciones de la Visión Artificial* [1] presso l'università di Cordoba in Spagna, per le applicazioni di Realtà Aumentata. Si basa esclusivamente su OpenCv e permette di individuare, all'interno di un'immagine, i classici marker bianchi e neri.
Il processo di rilevamento all'interno di ArUco è il seguente:

Applicazione di una procedura di *Thresholding adattivo*, necessaria ad ottenere i bordi all'interno del frame e per poter di conseguenza identificare i contorni.4.5
Thresholding adattivo
Durante il processo di *Thresholding* ogni pixel viene segnato come *object pixel*

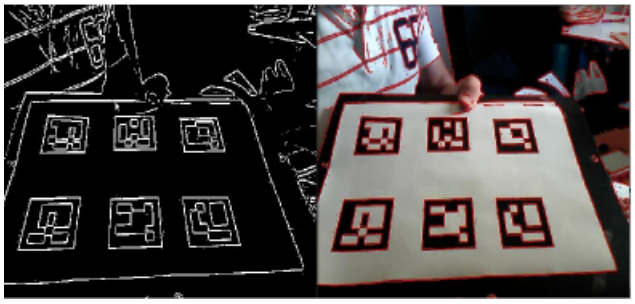


Fig. 4.5: ArUco: applicazione del thresholding adattivo al frame.

se il suo valore è più elevato di una certa soglia, mentre quelli nell'intorno hanno valori più bassi e vengono etichettati come *background pixel*. Questa convenzione è conosciuta come *threshold above*, esistono varie versioni della stessa procedura, per esempio *threshold below* che è l'opposto del precedente, *threshold inside* dove i pixel sono marcati come *object* se stanno all'interno di due valori di soglia e infine *threshold outside* che è la controparte della versione inside. Tipicamente si da un valore 1 ai pixel *object* e 0 ai pixel *background* e infine si colora l'immagine in bianco e nero a seconda dell'etichettatura.

Il punto chiave è la scelta del valore di soglia, ci sono molti modi per sceglierlo in maniera automatica (*automatic thresholding*). Si può scegliere il valore medio dei pixel, questa versione funziona bene su immagini senza rumore. Un approccio più sofisticato è quello di creare un istogramma delle intensità dei pixel e usare il punto più basso dell'istogramma, ma è di contro molto costoso. Un metodo relativamente semplice, che non richiede conoscienze specifiche dell'immagine ed è robusto al rumore è il seguente metodo iterativo:

- Viene scelto un valore iniziale di soglia T in maniera casuale.
- Vengono creati due insiemi insiemi di pixel(f(m,n) è il valore del pixel in posizione m,n):
  - $G_1 = f(m,n) : f(m,n) > T$ (object pixel)

- $G_2 = f(m,n) : f(m,n) \leq T$ (background pixel)

- vengono calcolati i valori medi di ogni insieme $m_1 e m_2$
- viene creato un nuovo valore di soglia $T' = (m_1 + m_2)/2$
- si riparte dal passo 2 finchè non otteniamo convergenza.

Il *Thresholding* viene chiamato adattivo quando vengono utilizzati differenti valori di T in differenti regioni dell'immagine.

A seguito della fase sopra descritta vengono identificati anche una serie di bordi indesiderati all'interno dell'immagine, oltre a quelli appartenenti ai marker reali. Il resto della procedura si occupa di scartare i bordi inutili, ovvero quelli con un piccolo numero di punti.
Viene applicata un'approssimazione poligonale dei contorni per estrarre quelli concavi con quattro spigoli come i rettangoli.
Si ordinano gli spigoli in direzione antioraria e successivamente si rimuovono i rettangoli troppo vicini 4.6.
Questo è richiesto perchè il processo di *Thresholding* adattivo normalmente rileva sia le parti interne che esterne dei bordi del marker, mentre di norma si vogliono conservare solo quelli esterni.

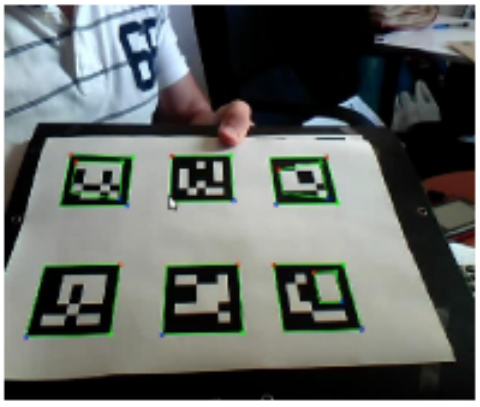


Fig. 4.6: ArUco: Eliminazione dei bordi interni rispetto quelli trovati nella fase di Thresholding.

Estratti i contorni del marker si passa alla sua identificazione. Si rimuove la proiezione prespettica per ottenere una visualizzazione frontale dell'area rettangolare usando la funzione di *homography*4.7. Si applica l'algoritmo di

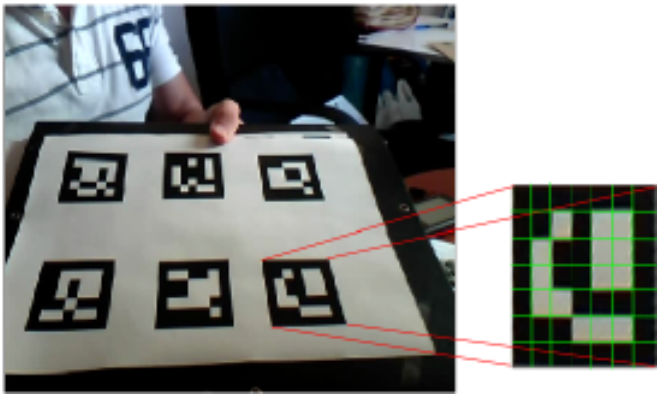


Fig. 4.7: ArUco: Eliminazione proiezione prospettica dal marker individuato nella scena.

Otsu per ottenere un'immagine solo bianca e nera (senza sfumature). Otsu è un metodo usato in computer vision per ottenere immagini binarie da immagini a livelli di grigio, che assume che l'immagine da elaborare contenga due classi di pixel(pixel background e pixel foreground) e quindi calcola il valore di soglia preferibile per separare le due classi.
Se è stato rilevato un marker, allora c'è un codice al suo interno. Il marker è diviso in una griglia 7x7, la cui parte interna 5x5 contiene le informazioni per l'id, il resto corrisponde al bordo esterno. Prima si controlla che il bordo esterno sia presente, dopodichè si legge la griglia interna di dimensione 5x5 controllando che fornisca un codice valido. Infine, se sono presenti i parametri della camera, vengono calcolati i parametri estrinseci del marker verso la camera.

Marker coding Ogni marker ha un codice interno dato da 5 parole di 5 bit ciascuna. La codifica applicata si basa sulla distanza Hamming. La differenza

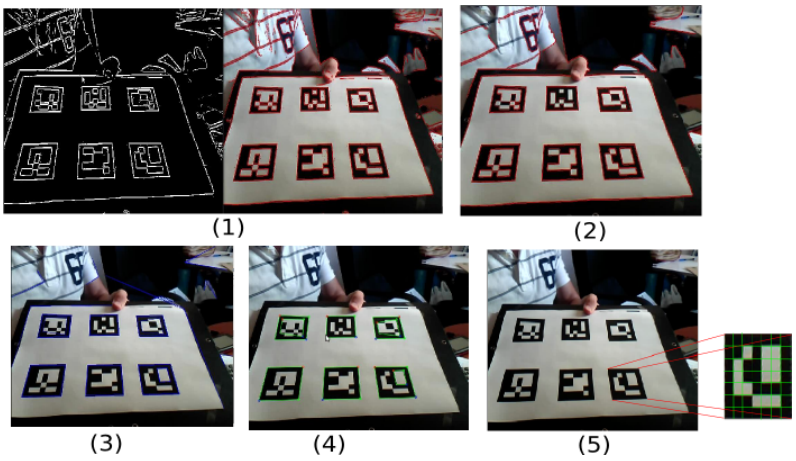


Fig. 4.8: Schema globale del processo di detection in ArUco:(1)Thresholding Adattivo,(2)approssimazione poligonale,(3)eliminazione poligoni con pochi punti,(4)eliminazione bordi interni,(5)eliminazione proiezione prospettica.

maggiore con la codifica di Hamming è che il primo bit viene sempre invertito, quindi l'id 0 che dovrebbe essere 00000 diventa 10000. L'idea è quella di prevenire un rettangolo completamente nero, riducendo così i falsi positivi scaturiti dagli oggetti presenti nell'ambiente analizzato. In totale ogni parola ha solo 2 bits di informazione rispetto i 5 implicati, gli altri 3 vengono utilizzati per il rilevamento degli errori. Come conseguenza si ottengono 1024 identificatori differenti(2 bit per ognuna delle 5 parole corrispondono a 10 bit complessivi e quindi a una rappresentazione massima di 1024 identificatori).

ArUco boards Il rilevamento di un singolo marker potrebbe fallire per differenti ragioni, come le condizioni di luce, i movimenti veloci della camera e le occlusioni. Per ovviare a questi problemi, ArUco fornisce lo strumento *Boards* 4.10. Una *Board* in *Augmented Reality* è un marker composto da più marker posizionati in una griglia.

Questa soluzione presenta due grossi vantaggi , il primo è la diminuzione della probabilità di non detection, in quanto con più marker è meno probabile

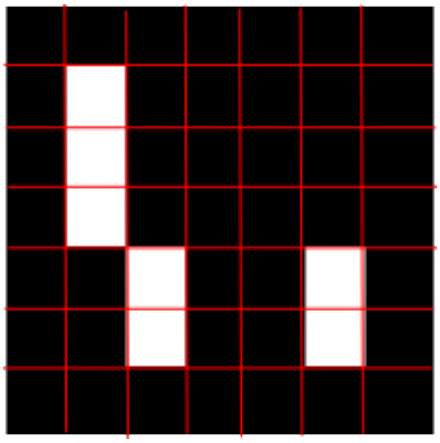


Fig. 4.9: Esempio di marker fiduciale in ArUco.

incorrere in errori dovuti alle condizioni di luce o di occlusione, mentre il secondo vantaggio è che con più marker si ottengono più punti per calcolare i parametri estrinseci della camera e quindi un incremento nell'accuratezza nella definizione del piano tridimensionale sul quale vengono renderizzate le informazioni 3D.

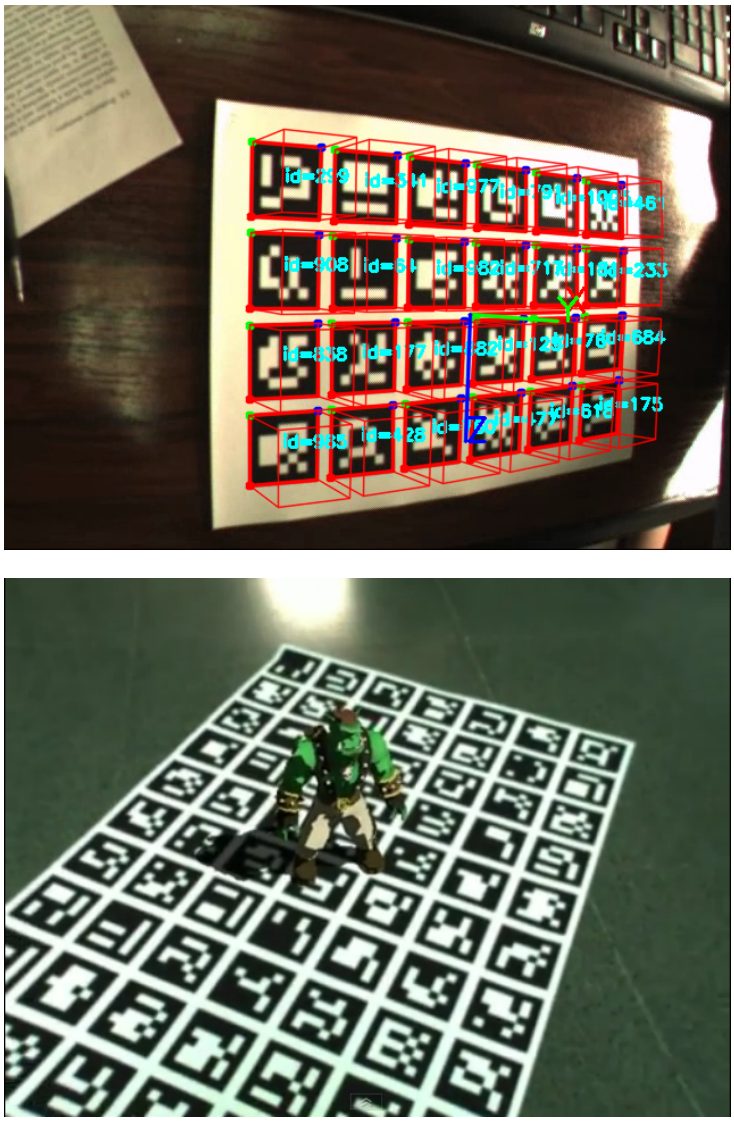


Fig. 4.10: Esempi di applicazione di boards in ArUco.

Capitolo 5

Implementazione di un Tracker di marker con caratteristiche naturali

In questo capitolo viene descritta la procedura di marker detection basata sulla libreria open-source ArUco e sulle estensioni implementate nel presente progetto. Viene descritta la parte fondamentale del motore di tracking e delle estensioni, fornendone le classi, gli attributi, i metodi e la loro spiegazione dettagliata.

5.1 Introduzione

Viene utilizzato il motore di tracking di ArUco per l'individuazione dei potenziali marker all'interno della scena. I marker potenziali sono rappresentati da poligoni convessi con 4 spigoli, ovvero quadrilateri con bordo nero. In ArUco i marker possibili diventano marker effettivi a fronte di un processo di verifica e riconoscimento dell'id interno, tramite la codifica di Hamming. In questa

tesi viene estesa l'idea di marker fiduciale definendo un nuovo tipo di marker con caratteristiche naturali. 5.1 L'idea generale del tracking implementato

Fig. 5.1: Marker Fiduciale e Marker Naturale

è quella di estrarre i pattern quadrati con bordo nero servendosi di Aruco, ottenendo i marker naturali potenziali e validarli tramite procedure di Image Matching 5.2.
I marker che passano la fase di validazione vengono etichettati con un identificatore univoco corrispondente a uno dei marker definiti in fase di inizializzazione. L'identificatore è necessario per la discriminazione delle informazioni virtuali da visualizzare.

5.1.1 Marker Naturale

Il *Marker Naturale* è un marker con bordo nero che contiene un'immagine con caratteristiche naturali come quella descritta in figura(5.1). Un' ulteriore differenza tra il marker naturale e quello fiduciale è che il primo necessita nell'angolo in alto a sinistra di un pattern bianco, indispensabile per fissa-

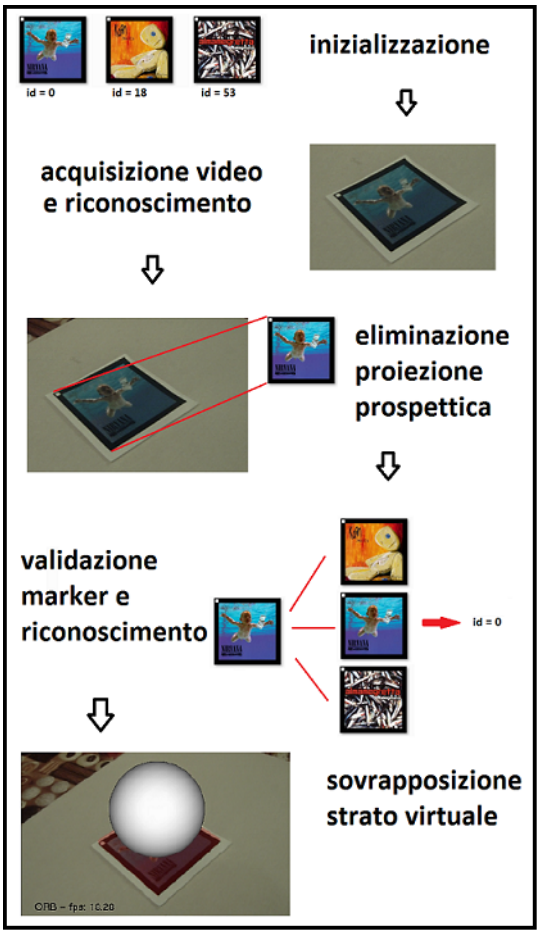


Fig. 5.2: Passi principali del Natural Features Tracker.

re l'orientamento degli oggetti tridimensionali, che vengono visualizzati sul piano identificato.

5.1.2 Strumenti

Gli strumenti utilizzati per il progetto descritto nel presente documento sono librerie opensource e il linguaccio C++:

- *OpenCV(Open Source Computer Vision)*: una libreria che offre funzionalità per compiti realtime di Computer Vision ([14] e [8]).

- *OpenGL(Open Source Graphic Library)*: libreria per lo sviluppo di applicazioni di grafica 3D e 2D([7]).
- *ArUco*: descritta in precedenza.

5.2 Implementazione base ArUco

Le classi fondamentali definite in ArUco, descritte in questa sezione ed ampliate nella successiva, sono:

- `FiducidalMarkers:` Classe che fornisce alcuni strumenti di utilità per la gestione dei marker fiduciali.
- `Marker:` Questa classe rappresenta un marker fiduciale.
- `MarkerDetector:` Classe principale per il processo di marker detection.
- `CameraParameters:` Parametri della camera.

5.2.1 Classe FiducidalMarkers

Di seguito viene fornita la spiegazione dei principali metodi della classe `FiducidalMarkers`.

- `static cv::Mat createMarkerImage(int id,int size)`: utilizzato per la creazione di un'immagine quadrata di dimensione data in input che rappresenta un marker AR identificato dal paramtro `id` specificato in input, utilizzando la versione modificata della codifica di Hamming. Esistono due tipi di marker fiduciali, quelli a 10 bit e quelli a 3 bit, questi ultimi supportati per applicazioni che necessitano un numero di marker non superiore a 7[2000 - 2006], mantre i primi ricoprono un range di identificatori da 0 a 1023.

- `static int detect(const cv::Mat &in,int &nRotations)` : Il potenziale marker estratto dal frame e privo di proiezione prospettica(quindi in vista frontale) viene passato in iput a questo metodo, che ne stabilisce l' eventuale validità. Restituisce l'id del marker individuato oppure -1 nel caso in cui il marker non sia valido. Stabilisce, inoltre, il numero di rotazioni di 90 gradi in direzione oraria necessarie a settare il marker in posizione canonica. Per effettuare le operazioni sopra descritte il metodo applica un filtro binario all'immagine in input e tramite la codifica di Hamming ne analizza il contenuto ricavando il numero di rotazioni calcolando la distanza di Hamming delle 5 parole binarie da quelle plausibili.

5.2.2 Classe Marker

La classe `Markers` estende una struttura dati di tipo `std::vector<cv::¬ Point2f>`. Un marker può essere visto come un insieme di 4 punti nello spazio bidimensionale nelle coordinate della finestra.

Attributi

- `int id`: attributo di tipo intero per l'identificazione univoca del marker.
- `float ssize`: grandezza dei lati del marker in metri.
- `cv::Mat Rvec, Tvec`: vettori di rotazione e traslazione tridimensionale rispetto la camera.

Costruttori

- `Marker()`: Costruttore senza parametri per l'istanziazione di un marker con attributi fittizi.

- `Marker(const Marker &M)`: Costruttore per l'instanziazione di un marker copia di quello in input.

- `Marker(const std::vector<cv::Point2f> &corners,int _id=-1)`: Costruttore per l'istanziazione di un marker con spigoli e identificatore dato in input.

Metodi

- `bool isValid()`: indica se l'oggetto è valido.

- `void draw(cv::Mat &in, cv::Scalar color, int lineWidth=1¬ , bool writeId=true)`: metodo di utilità per il disegno dei bordi e dell'identificatore al centro del marker individuato all'interno del frame.

- `void calculateExtrinsics(float markerSize,const CameraParameters &CP)`: Calcola i parametri estrinseci, quindi vettore di rotazione e di traslazione rispetto la camera, prendendo in input la grandezza del lato del marker in metri.

- `void glGetModelViewMatrix(double modelview_matrix[16])`: Dati i parametri estrinseci della camera, restituisce la matrice `GL_MODELVIEW` di OpenGL. Settando questa matrice viene settato il sistema di riferimento del marker.

- `cv::Point2f getCenter()`: Restituisce il centro del marker.

- `float getPerimeter()`: Restituisce il perimetro del marker.

- `float getArea()`: Restituisce l'area del marker.

5.2.3 Classe MarkerDetector

`MarkerDetector` è la classe che rappresenta il motore di tracking di ArUco, fornisce le funzionalità per individuare i marker all'interno della scena, e la possibilità di configurare alcune caratteristiche del processo di tracking come l'algoritmo utilizzabile per effettuare il *thresholding* del frame.

- `enum CornerRefinementMethod {NONE,HARRIS,SUBPIX}`: insieme dei metodi di corner detection applicabili, Harris è quello di default.

- `enum ThresholdMethods {FIXED_THRES,ADPT_THRES,CANNY}`: insieme dei metodi di *thresholding* applicabili, quello adattivo è il metodo di default.

Costruttori

- `MarkerDetector()`: Costruttore per l'istanziazione di un marker detector con parametri settati ai valori di default.

Metodi

- `void detect(const cv::Mat &input, std::vector<Marker> &detectedMarkers, CameraParameters camParams, float markerSizeMeters=-1)`: Rileva i marker nell'immagine passata, se vengono fornite le informazioni relative ai parametri della camera e alla grandezza del marker vengono calcolati anche i parametri estrinseci dei marker rilevati.

- `void setThresholdMethod(ThresholdMethods m)`: metodo *setter* della funzione di *thresholding*.

- `ThresholdMethods getThresholdMethod()`: restituisce il metodo di threshold utilizzato nel motore di detection.

- `void setThresholdParams(double param1, double param2)`:setta i parametri per la funzione di *threshold* che corrispondono rispettivamente alla grandezza dell'area attorno al pixel per calcolare il valore di soglia per lo stesso e alla costante sottratta dalla media.

- `void getThresholdParams(double &param1, double &param2)`: restituisce i parametri di cui sopra.

- `const cv::Mat & getThresholdedImage()`: restituisce un riferimento all'immagine interna alla quale è stata applicata la procedura di threshold.

- `void setCornerRefinementMethod(CornerRefinementMethod method¬ )`: setta il metodo per la procedura di corner detection.

- `CornerRefinementMethod getCornerRefinementMethod()`: restituisce il metodo di cui sopra.

- `void setDesiredSpeed(int val)`: Specifica un valore che indichi la velocità di detection richiesta. Può assumere valori da 0 a 3. Con una velocità alta viene impiegato `setCornerRefinementMethod(NONE)` e si utilizza una dimensione dell'immagine che rapresenta il marker (a fronte dell'eliminazione della proiezione prospettica) più piccola. In modalità lenta si usa `setCornerRefinementMethod(HARRIS)` e un'immagine che rappresenti il marker più grande.

- `int getDesiredSpeed()`: restituisce il valore associato alla velocità.

- `void thresHold(int method, const cv::Mat &grey, cv::Mat &¬ thresImg, double param1, double param2)`: applica la procedura di soglia con i parametri specificati in input all'immagine passata.

- `void detectRectangles(const cv::Mat &thresImg, vector<std¬ ::vector<cv::Point2f>> & candidates)`: si occupa del rilevamento dei marker candidati(rettangoli) trovati nell'immagine passata dal processo di Threshold.

- `vector<std::vector<cv::Point2f>> &getCandidates()`: Restituisce la lista di candidati per i quali non è stato individuato alcun identificatore.

- `void warp(cv::Mat &in, cv::Mat &out, cv::Size size,\ std¬ ::vector<cv::Point2f> points)`: Prende l'immagine in input che contiene il marker con corner indicati nel parametro `points` e ne restituisce una con il marker nella posizione canonica in vista frontale di dimensione `size`.

5.2.4 Classe CameraParmaters

`CameraParameters` definisce i parametri utili per la camera, distorsione, matrice della camera, dimensione dell'immagine ripresa.

Attributi

- `cv::Mat CameraMatrix`: rappresenta la matrice che ingloba i parametri intrinseci lunghezza focale delle lenti su entrambi gli assi(f_x, f_y) espressi in pixel e il centro ottico del sensore in pixel(c_x, c_y).

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- `cv::Mat Distorsion`: vettore che ingloba i parametri intrinseci rappresentati dai coefficienti di distorsione tangenziale e radiale $[k_1, k_2, p_1, p_2]$
- `cv::Size CamSize`: dimensioni del frame proveniente dalla camera.

Costruttori

- `CameraParameters()`: Costruttore senza parametri.
- `CameraParameters(cv::Mat cameraMatrix, cv::Mat distorsionCoeff, cv::Size size)`: Costruttore per l'instanziazione della camera con parametri instrinseci passati in ingresso.
- `CameraParameters(const CameraParameters &CI)`: Costruttore copia.

Metodi

- `void setParams(cv::Mat cameraMatrix, cv::Mat distorsionCoeff, cv::Size size)`: setter dei parametri principali.
- `bool isValid()`: indica se l'oggetto è valido controllando che i parametri principali siano settati.
- `void readFromFile(string path)`: legge i parametri della camera da un file salvato con il metodo `saveToFile`.
- `void saveToFile(string path)`: salva i parmetri della camera sul file indicato in input.

- `void readFromXMLFile(string filePath)`: legge i parametri da un file YAML generato con l'utility di calibrazione di opencv2.2.
- `void resize(cv::Size size)`: Adatta i parametri della camera a seconda delle dimensioni indicate, utilizzato durante il resize della finestra.
- `void glGetProjectionMatrix( cv::Size orgImgSize, cv::Size size, double proj_matrix[16], double gnear, double gfar, bool invert=false)`: Dati i parametri intrinseci, il metodo restituisce la matrice `GL_PROJECTION` OpenGL.

5.3 Estensione per il tracking di marker con caratteristiche naturali

Sono state apportate modifiche puntuali alla libreria ArUco per estenderne le funzionalità ed ottenere un tracker di marker con caratteristiche naturali. Le modifiche principali riguardano la fase di identificazione del marker, dove è stata sostituita la codifica di Hamming con procedure di Image Matching. Nel seguito verranno descritte le estensioni alle classi esistenti e alcune procedure di utilità sviluppate per il task.

5.3.1 Classe NaturalMarkers

La classe `NaturalMarkers` estende `FiducialMarkers` aggiungendo un unico metodo per la validazione del marker stesso:

- `void detectForNaturalFeatures(const cv::Mat &in, int &nRotations)`: è il corrispettivo del metodo `detect(const cv::Mat &in¬ ,int &nRotations)` per i marker fiduciali, si occupa di controllare la

validità del marker. L'immagine passata in ingresso, priva di proiezione prospettica, deve avere un bordo nero e un quadrato bianco in uno degli angoli. Fissando per convenzione come corner bianco quello in alto a sinistra, l'immagine viene analizzata i tutti e quattro i *corners* finchè non viene trovato il pattern bianco, il numero di rotazioni di 90 gradi in direzione oraria necessarie per trovarlo viene restituito in `nRotations` 5.3. Se il corner bianco non viene individuato il marker non è corretto e `nRotations = -1`. La procedura di attribuzione id viene effettuata successivamente, per ora basti sapere che il marker sia candidabile in quanto provvisto di bordo nero e pattern bianco in uno degli angoli.

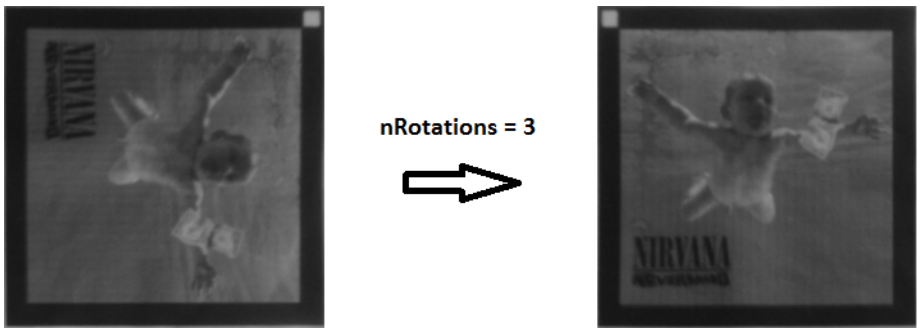


Fig. 5.3: Rotazione del marker.

5.3.2 Classe Marker e CameraParameters

La classe `Marker` non ha bisogno di sviluppi ulteriori in quanto un *marker naturale* ha le stesse caratteristiche del marker fiduciale, quindi 4 corner, un identificatore univoco e dei vettori di rotazione e traslazione per i cambi nel sistema di riferimento della camera.

Stesso discorso vale per `CameraParameters` che include tutte le funzionalità necessarie per rappresentare la camera nella versione base e che quindi non necessità di modifiche.

5.3.3 Classe MarkerDetector

`Markerdetector` viene ampliata con le seguenti aggiunte:

Attributi

- `enum NaturalFeaturesMethods {WNF_NONE, ORB, SURF, SIFT, BRIEF¬ }`: insieme degli algoritmi supportati e utilizzabili nel processo di *image matching* e quindi identificazione del marker.

Costruttori

- `MarkerDetector(NaturalFeaturesMethods detection_approach)`: costruttore per l'instanziazione di un detector che supporti il tracking di marker naturali. Setta i parametri di default come i costruttori descritti nel capitolo precedente, tranne che per la grandezza dell'immagine da passare al processo di image matching, in quanto i valori utilizzati nella versione base sono troppo piccoli perchè l'algoritmo di image matching funzioni bene.

Metodi

- `NaturalFeaturesMethods getNaturalFeaturesMethod()`: Restituisce il valore associato al metodo utilizzato per l'identificazione del marker.

- `void setNaturalFeaturesMethod(NaturalFeaturesMethods)`: setta il valore di cui sopra.

- `char* getNaturalFeaturesType()`: restituisce la stringa che rappresenta il nome dell'algoritmo utilizzato nel identificazione del marker.

- `void detectWithNaturalFeatures(const cv::Mat &input,std::¬ vector<Marker> &detectedMarkers, CameraParameters camParams¬ ,float markerSizeMeters=-1)`: prende in input il frame proveniente dalla camera ed eventualmente costruisce un vettore di marker naturali. Ogni marker ha associato l'identificativo risultante dal processo di identificazione che viene fatto all'interno del metodo e se vengono forniti i parametri della camera vengono calcolati anche i parametri estrinseci dei marker relativi alla camera.

Fig. 5.4: Processo di detection.

5.3.4 NaturalFeaturesTrackingUtils

Il processo di estrazione dei marker implementato nella funzione `void detectWithNaturalFeatures(...)` utilizza una procedura di image matching per l'assegnazione di un identificatore a ognuno dei marker che risulteranno validi. I marker che superano la prima fase di validazione, ovvero quella che controlla la presenza di un bordo nero e di un quadrato bianco in uno dei

corner dell'immagine, vengono sottoposti a questo processo e se viene trovata una corrispondenza con i marker definiti in fase di inizializzazione viene assegnato un id corrispondente, altrimenti il marker viene scartato.

Nel processo di identificazione implementato nelle funzioni di utilità di `NaturalFeaturesTrackingUtils` vengono utilizzate alcune funzioni di estrazione e descrizione di punti caratteristici della libreria OpenCV. Gli strumenti essenziali sono suddivisibili in tre categorie *Detector, Extractor e Matcher* che ognuno degli approcci supportati nella tesi implementa esponendo un interfaccia da utilizzare. Il Detector si occupa di individuare all'interno dell'immagine i punti caratteristici, l'Extractor fornisce una descrizione secondo le procedure dei capitoli 2 e 3 ai punti individuati, mentre il Matcher confronta i descrittori calcolati su due immagini e restituisce una lista di possibili match effettuati con successo. Per ognuno degli algoritmi supportati OpenCV fornisce i seguenti strumenti

- Detector: `OrbFeatureDetector, SurfFeatureDetector, SiftFeatureDetector`(Per BRIEF si utilizza un Detector di tipo SURF).
- Extractor: `OrbDescriptorExtractor, SurfDescriptorExtractor¬ , SiftDescriptorExtractor, BriefDescriptorExtractor`.
- Matcher: `BruteForceMatcher, FlannBasedMatcher...`

Di seguito viene presentato un esempio di codice utilizzato per il confronto, con ORB, tra due immagini generiche. Il codice è semplificato al massimo per poterne rendere semplice la comprensione. Si occupa di individuare i punti caratteristici di ogni immagine passata in ingresso, per ognuno costruirne una descrizione e individuare le corrispondenze migliori tra le descrizioni fornite.

```
vector<DMatch> ImageMatchingORB(IplImage* image0,
 IplImage* image1){
    //Estrazione punti caratteristici
  OrbFeatureDetector detector(100);
  vector<KeyPoint> keypoints0, keypoints1;
  detector.detect(image0, keypoints0);
    detector.detect(image1, keypoints1);
    //Descrizione punti caratteristici
  OrbDescriptorExtractor extractor;
  Mat descriptors0, descriptors1;
  extractor.compute(image0, keypoints0, descriptors0);
  extractor.compute(image1, keypoints1, descriptors1);
    //Matching fra descrittori
  BruteForceMatcher<HammingLUT> matcher;
  std::vector<DMatch> matches;
  matcher.match(descriptors0, descriptors1, matches);
  return matches;
}
```

L'oggetto di tipo `DMatch`(documentato in [4] e [5])contiene al suo interno i seguenti attributi:

- `int queryIdx`: posizione all'interno del primo vettore di descrittori del primo elemento facente parte della coppia
- `int trainIdx`: posizione all'interno del secondo vettore di descrittori del secondo elemento facente parte della coppia
- `float distance`: valore che rappresenta la distanza tra la prima descrizione e la seconda (nel caso migliore 0).

Nell'algoritmo della tesi viene applicato un passaggio ulteriore per aumentare la precisione nell'identificazione. Viene sostituito il metodo

```
void DescriptorMatcher::match(const Mat& queryDescriptors, const
    Mat& trainDescriptors, vector<DMatch>& matches);
```

che si limita a trovare il migliore corrispettivo per ogni descrittore del primo insieme rispetto quelli del secondo, con il metodo

```
void DescriptorMatcher::knnMatch(const Mat& queryDescriptors,
   const Mat& trainDescriptors, vector<vector<DMatch> >& matches
   , int k);
```

che, invece, estrae per ogni descrittore del primo insieme i migliori k confronti(matches), se esistono, rispetto quelli del secondo insieme. Il passaggio ulteriore viene applicato per evitare i falsi positivi, questo perchè possono esistere descrizioni che si avvicinano ad altre, ma che non rappresentano lo stesso punto caratteristico. Per ridurre al massimo queste situazioni di ambiguità e non incidere in maniera negativa sul tempo calcolo si utilizza un valore di k uguale a 2 e si considerano come confronti corretti soltanto quelli i cui due valori di distanza rispettano la seguente formula:

$$\frac{d_1}{d_2} < nndr$$

dove d_1 è il valore di distanza della *prima scelta*, d_2 quello della seconda e $nndr = 0.6$ è la costante che rappresenta *Nearest Neighbor Distance Ratio* ovvero il rapporto tra le distanze di due corrispondenze vicine. In sintesi la corrispondenza trovata viene mantenuta soltato se ha un valore di distanza minore del valore della seconda per una costante moltiplicativa $nndr$, quindi se la seconda è *abbastanza lontana* da poter considerare la corrispondenza biunivoca e non ambigua. All'interno `NaturalFeaturesTrackingUtils` vengono fornite le seguenti procedure:

- `void ORBfindObjectKPDescr()`: funzione di utilità per l'estrazione di punti caratteristici con ORB dalle immagini di riferimento durante l'inizializzazione del sistema. Le funzioni equivalenti per gli altri approcci supportati dal sistema sono:
  - `void SURFfindObjectKPDescr()`
  - `void SIFTfindObjectKPDescr()`
  - `void BRIEFfindObjectKPDescr()`
- `int DetectORB(IplImage* marker_image)`: funzione di utilità per l'analisi del marker estratto dal frame e individuazione dell'id corrispondente. Per quanto riguarda gli altri approcci vengono fornite le seguenti funzioni:
  - `int DetectSURF(IplImage* marker_image)`
  - `int DetectSIFT(IplImage* marker_image)`
  - `int DetectBRIEF(IplImage* marker_image)`

Capitolo 6

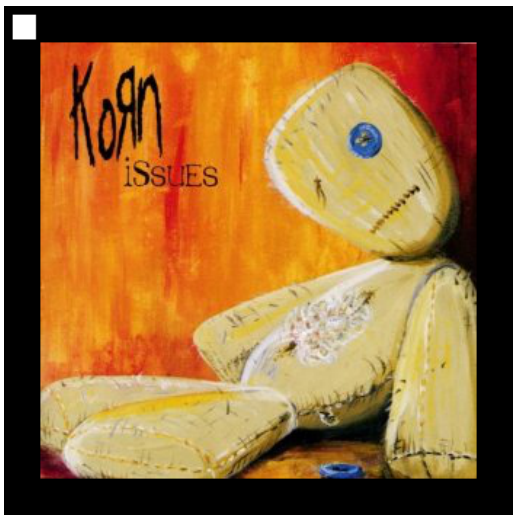
Test comparativi e analisi delle prestazioni

Nel presente capitolo vengono confrontati i quattro approcci utilizzati nell'implementazione del tracker creato in questo lavoro di tesi.
Gli algoritmi usati sono SIFT, SURF, BRIEF e ORB e per ognuno sono state analizzate le prestazioni sul campo di applicazione reale, rilevando e confrontando i tempi di calcolo e valutando la correttezza. Infine è stata fornita una comparazione tra il sistema implementato e quello nativo di ArUco in termini di frame processati per secondo e di precisione.

6.1 Test effettuati

Per effettuare i test, descritti in seguito, sono state utilizzate le seguenti immagini come marker da rilevare all'interno del sistema. I marker, come discusso in precedenza, hanno il tipico bordo nero e il corner top - left bianco.
I test effettuati sono nell'ordine:

(a) marker1



(b) marker2

(c) marker3

Fig. 6.1: Marker utilizzati

- Tempi di estrazione delle features e numero features sulle immagini di riferimento con ognuno dei 4 approcci.
- Un marker nella scena:
  - Tempi di estrazione delle features con ognuno dei 4 approcci.
  - Tempi di descrizione delle features con ognuno dei 4 approcci.

  - Correct matches con ognuno dei 4 approcci.
- Tempi di estrazione e descrizione con due marker nella scena.
- Tempi di estrazione e descrizione con tre marker nella scena.
- Test comparativo sul framerate di SURF, SIFT, BRIEF e ORB.
- Test comparativo sul framerate di ORB e implementazione originale di ArUco.

6.2 Tempi di estrazione e numero dei punti caratteristici estratti dalle immagini di riferimento

Per ognuno dei tre marker da rilevare durante il tracking è stato calcolato il tempo di estrazione e il numero di features estratte ottenendo i risultati mostrati in figura 6.2, 6.3, 6.4. Si può notare da6.4 che il marker 3 ha il numero più alto di features estratte. Questo implica, come vedremo in seguito, un notevole aumento nei tempi di detection in tutti gli approcci. A tal proposito è bene precisare che con ORB si ha la possibilità di imporre un tetto massimo di punti caratterisitici da estrarre, questo agevola il funzionamento in tempo reale del sistema.

L'aumento nei tempi di tracking, oltre che al calcolo delle features, va in parte imputato anche alla fase di confronto (necessario ad individuare la corrispondenza tra marker rilevato e marker di riferimento) tra le features delle immagini di riferimento e immagine estratta dal frame video.
Più alto è il numero di features da confrontare più sarà il tempo necessario ad effettuare il calcolo e individuare il marker.

L'unico punto di forza di ORB non è quello di poter limitare il numero di features estratte e per dimostrarlo praticamente è stato inserito nel test effettuato anche la versione di ORB con una configurazione relativa il numero di features massime da estrarre uguale a 700. A fini statistici sono stati confrontati i tempi tra ORB(700) e gli altri approcci e si è rielvato che i tempi sono paragonabili a ORB(100) quindi molto bassi, nonostante il numero elevato di features estratto.

Nei test successivi si è utilizzato ORB con soglia 100 perchè, come discusso, fornendo le 100 features migliori permette di coniugare velocità (nel calcolo delle features e nel matching tra esse) e potere distintivo nella rappresentazione.

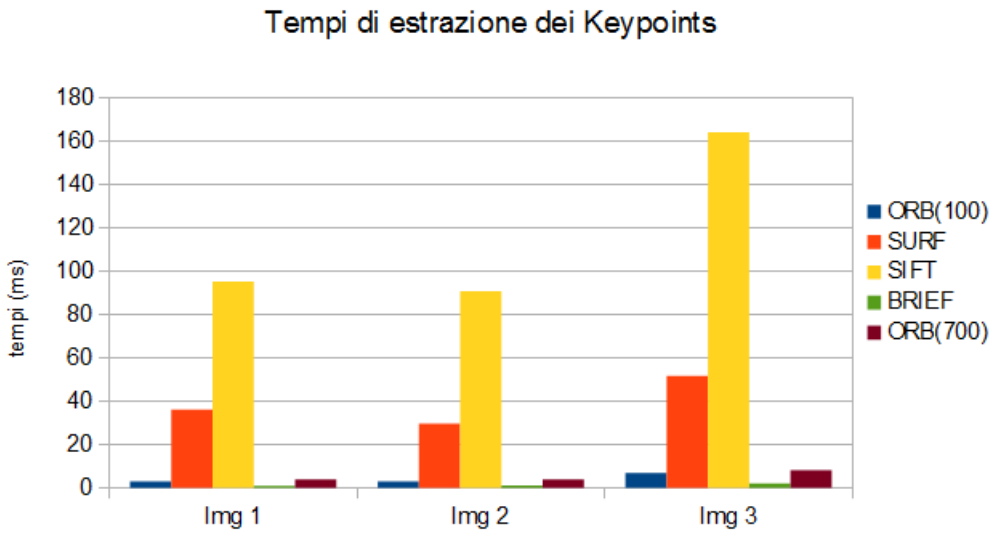


Fig. 6.2: Risultati Test1 (Fase di inizializzazione): Tempi di estrazione dei punti caratteristici sulle tre immagini campione per ognuno dei quattro approcci.

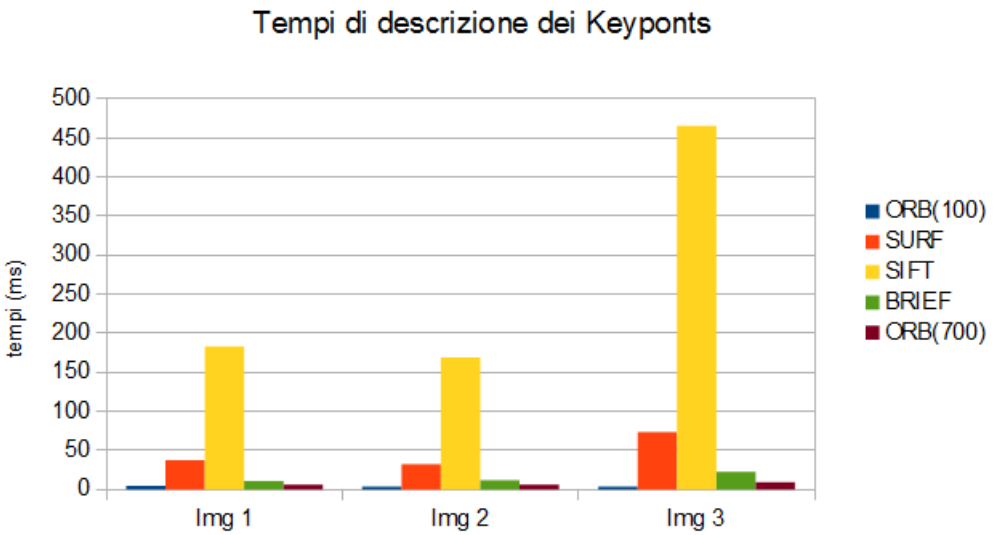


Fig. 6.3: Risultati Test1 (Fase di inizializzazione): Tempi di descrizione dei punti caratteristici individuati sulle tre immagini campione per ognuno dei quattro algoritmi.

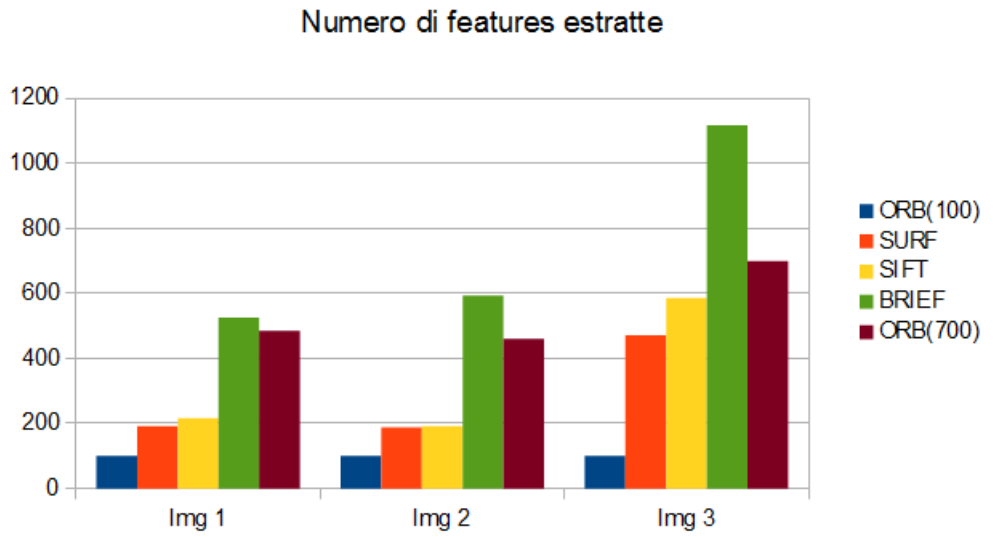


Fig. 6.4: Risultati Test1 (Fase di inizializzazione): Numero Features estratte sulle tre immagini di riferimento per ognuno degli algoritmi usati.

6.3 Velocità e correttezza con un unico marker sulla scena

Il secondo test è stato effettuato sulla scena, quindi sui frame della camera. Per ognuno dei quattro algoritmi sono state valutate le performance in termini di velocità di estrazione e confronti corretti. Nel test è stato ripreso un solo marker all'interno della scena. Anche in questo caso ORB(100) fornisce le migliori risposte in quanto a velocità di calcolo e precisione (valutata come percentuale tra features estratte e features che hanno un corrispettivo nelle immagine di riferimento). Si pone attenzione sul fatto che BRIEF, nonostante la sua velocità nel calcolo delle features, degrada nella fase di confronto a causa del numero inutilmente elevato di features estratte. A conferma del fatto che BRIEF dia pessime prestazioni anche a livello di precisione, si può vedere quanto sia bassa la percentuale di confronti corretti. Questo accade perchè BRIEF, come discusso nel capitolo 3, non fornisce invarianza di scala e di rotazione e spesso può capitare che l'immagine rilevata sia ruotata rispetto a quella di riferimento e sicuramente scalata. SIFT e SURF sono entrambi precisi nella descrizione, ma allo stesso tempo lenti (SIFT molto più di SURF) nella estrazione e nel confronto, questo li rende sicuramente molto validi per task che non richiedano computazioni realtime e che di contro necessitino di un elevato potere distintivo.

6.4 Velocità di detection con due e tre marker nella scena

Nel test 3 6.8 si è voluta analizzare la velocità di detection inserendo nella scena due marker e per completezza è stata effettuata la stessa misura con

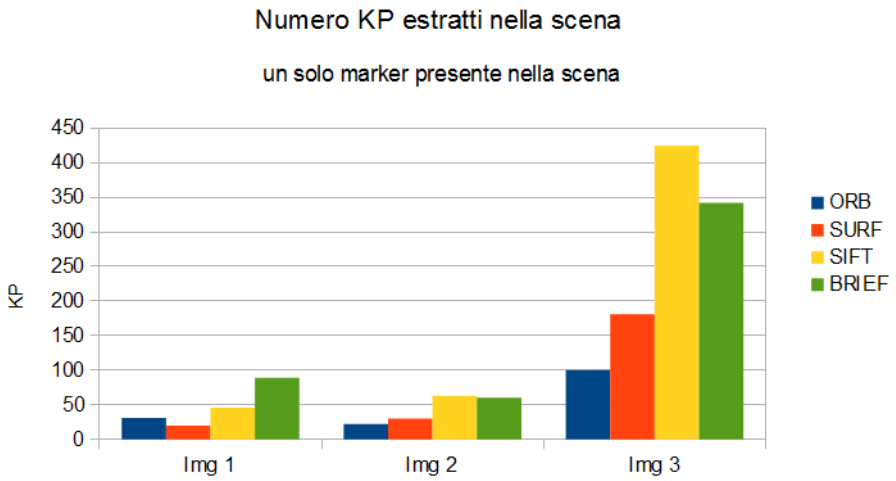


Fig. 6.5: Risultati Test2: Numero di KeyPoints individuati dai vari algoritmi su ognuna delle tre immagini campione estratte dalla scena.

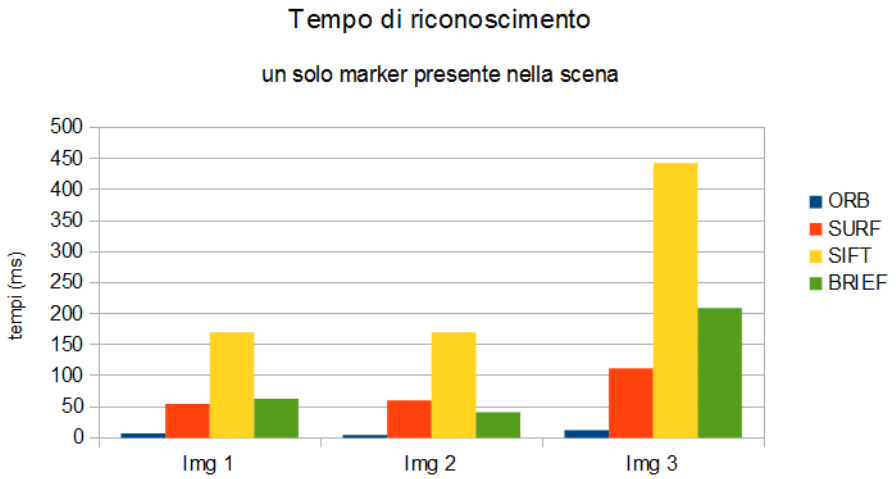


Fig. 6.6: Risultati Test2: Tempi impiegati dagli algoritmi per l'estrazione dei marker e il riconoscimento delle immagini campione all'interno della scena.

tutti e tre i marker nella scena 6.9. In questo test è visibile come ORB rispetto agli altri approcci permetta di ottenere risposte in tempo reale.

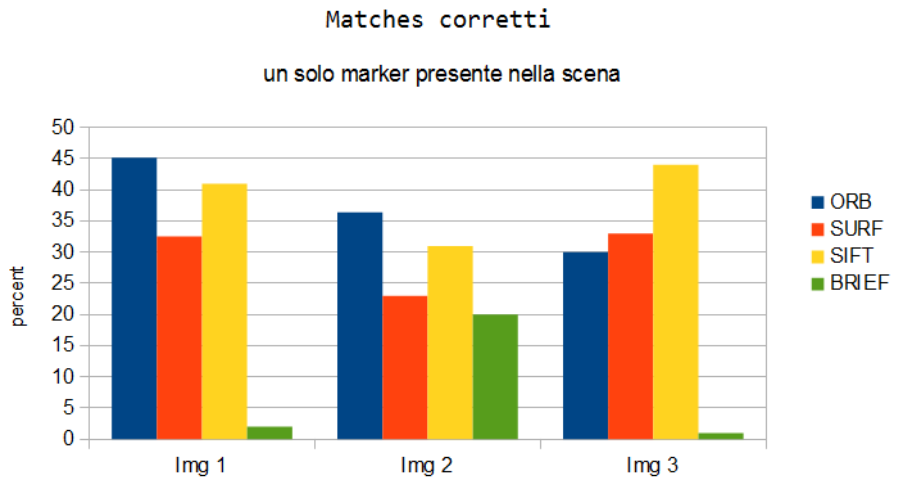


Fig. 6.7: Risultati Test2: Numero di confronti corretti tra i punti caratteristici individuati dai vari algoritmi sulle tre immagini campione estratte dalla scena e quelli estratti in fase di inizializzazione.

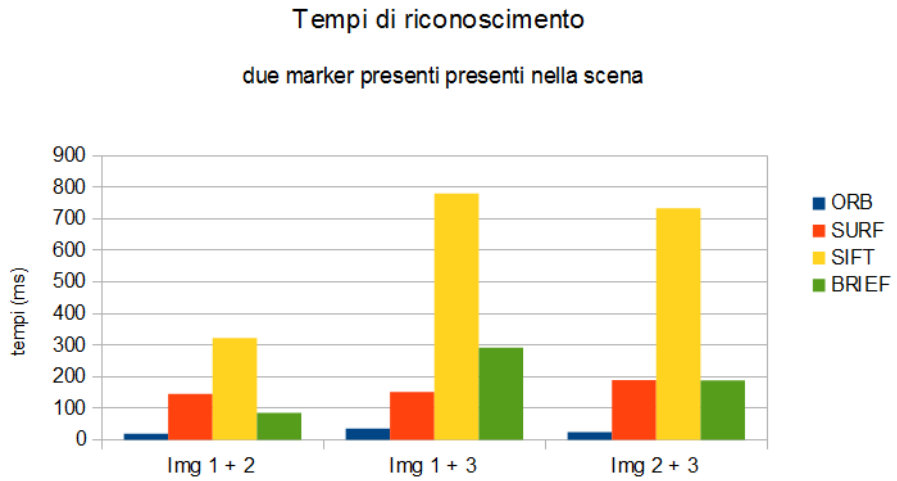


Fig. 6.8: Risultati Test3: Tempi di estrazione e riconoscimento di due marker all'interno della scena

6.5 Frame per secondo per ognuno dei quattro algoritmi

In questo test è stato analizzato il comportamento in termini di frame per secondo dei 4 algoritmi utilizzati. Per eseguire il test è stato registrato un video, che riprendesse i marker in diverse condizioni di luce, rotazione, scalatura,

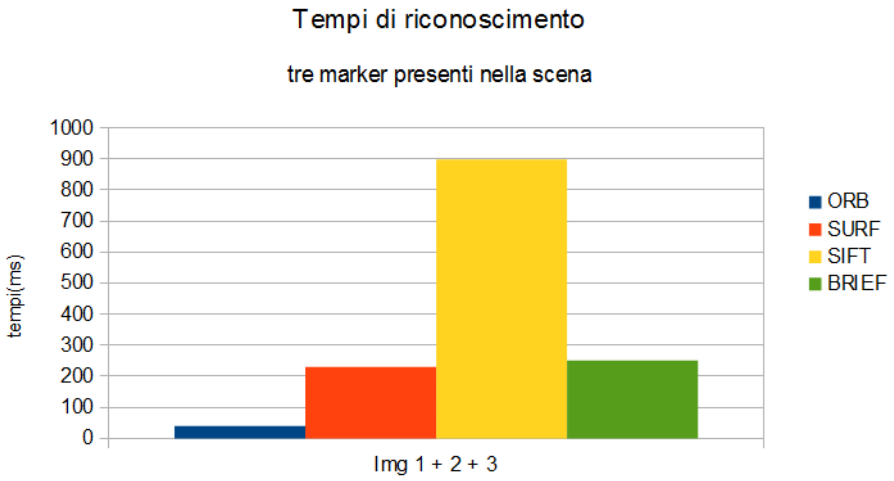


Fig. 6.9: Risultati Test4:: Tempi di estrazione e riconoscimento di tre marker all'interno della scena

cambi di punti di vista e occlusione, usandolo come input del meccanismo di rilevamento implementato, per analizzarne il comportamento sulla base degli approcci utilizzati.

Analisi del tracking con caratteristiche naturali SIFT ha le prestazioni peggiori in termini di velocità di calcolo, difatti il framerate si aggira attorno ai 6 frame per secondo evidenziando la lentezza dell'approccio rispetto agli altri studiati. 6.10

Come descritto nel terzo capitolo SURF migliora le prestazioni in termini di rapidità rispetto SIFT, ma rimane comunque oneroso rispetto ad altri.6.12 In entrambi si nota la precisione del calcolo, in quanto l'ID del marker da rilevare corrisponde a quello associato alla figura CUBO, ma è altrettanto visibile la scarsa velocità dal framerate rilevato. Al contrario BRIEF 6.13 ha una velocità comparabile a ORB, ma pecca in precisione e potenza descrittiva dovuta ai difetti insiti nell'invarianza di scala e rotazione dell' algoritmo, si nota infatti dall'immagine che il marker associato al CUBO viene erroneamente scambiato con quello associato al TEIERA. In conclusione, ORB 6.14fornisce il miglior compromesso tra velocità e precisione di calcolo, offrendo un

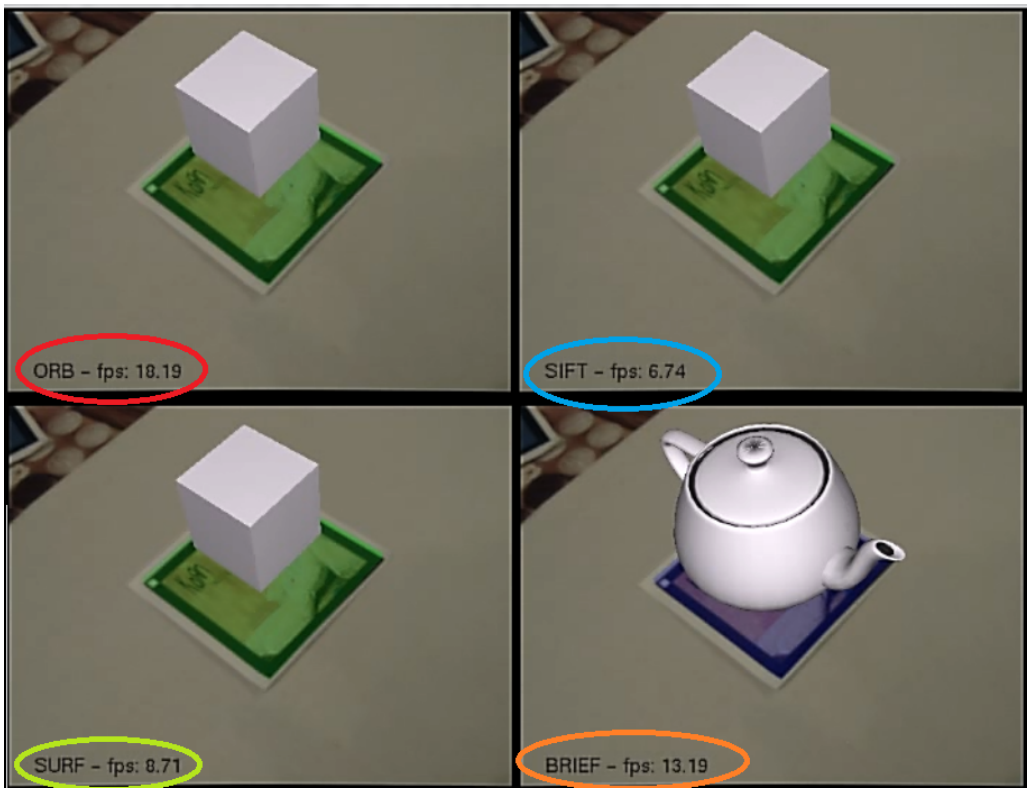


Fig. 6.10: Comparazione in termi di prestazioni e frame per secondo dei quattro algoritmi sulla scena. Vengono evidenziati i valori del framerate.

framerate vicino ai 20 fps, rimanendo efficiente nella potenza descrittiva.

6.6 Comparazione in termini di frame per secondo tra ORB e ArUco

L'ultimo test mette a confronto l'algoritmo implementato in questo lavoro con l'algoritmo sul quale si basa il lavoro descritto in questo documento. Sono stati registrati due video in condizioni molto simili, praticamente identiche.

Nel primo video viene rappresentata una scena contenente tre marker di tipo fiducial bianchi, mentre nel secondo sono stati ripresi i marker con caratteri-

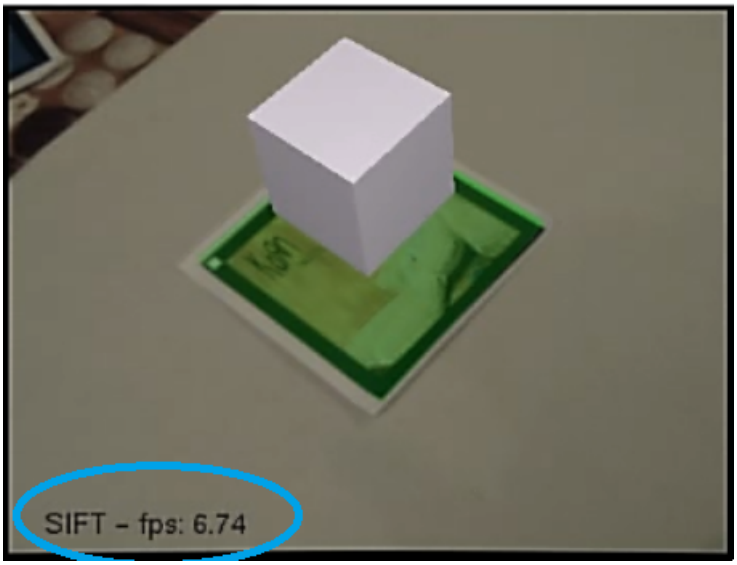


Fig. 6.11: Prestazioni dell'architettura inizializzata con algoritmo SIFT in termini di framerate sulla scena.

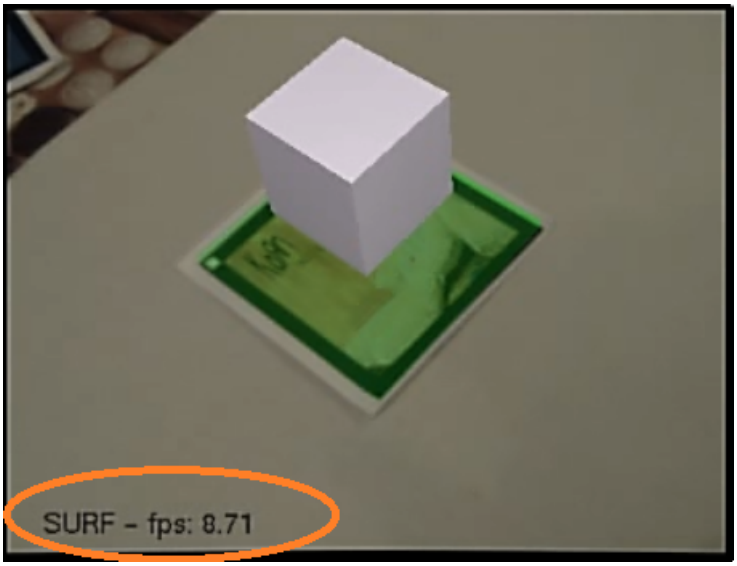


Fig. 6.12: Prestazioni dell'architettura inizializzata con algoritmo SURF in termini di framerate sulla scena.

stiche naturali. I video sono stati sottoposti rispettivamente alla procedura di detection di ArUco e alla procedura implementata basata sull'approccio ORB. 6.15. Il vantaggio di prestazioni di ArUco esiste, ma utilizzando ORB

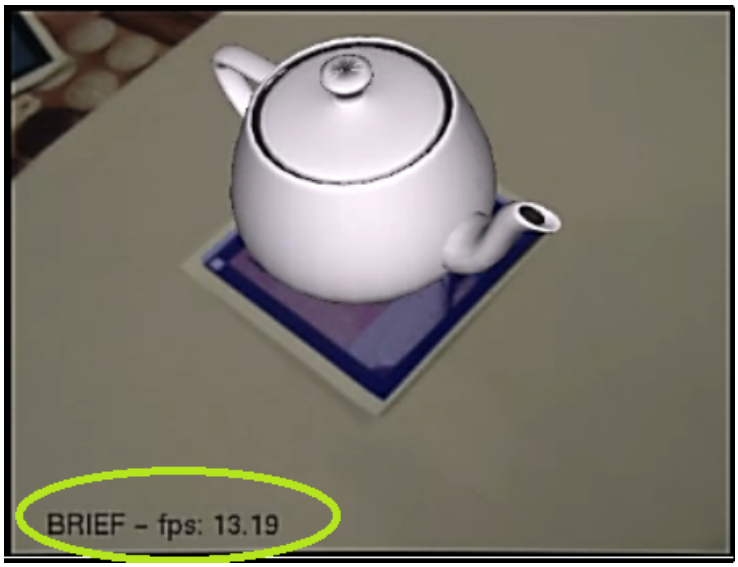


Fig. 6.13: Prestazioni dell'architettura inizializzata con algoritmo BRIEF in termini di framerate sulla scena.

Fig. 6.14: Prestazioni dell'architettura inizializzata con algoritmo ORB in termini di framerate sulla scena.

non è così evidente. Con un solo marker nella scena le prestazioni sono simili, anche se leggermente a favore di ArUco. Il motivo della migliore prestazione dell'approccio ArUco classico va cercato nel fatto che con ArUco il processo

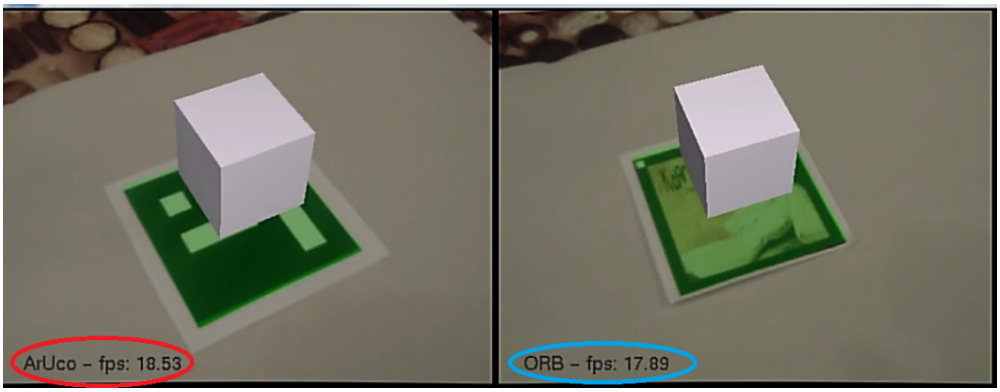


Fig. 6.15: Confronto delle prestazioni tra l'architettura di base ArUco e l'architettura estesa con algoritmo ORB in termini di frame per secondo.

di identificazione del marker consta di un rapido confronto bit a bit, mentre su marker con caratteristiche di tipo naturale il processo di identificazione consta di più fasi dispendiose come l'estrazione dei keypoints, la descrizione e il matching.

La differenza è ancora più evidente quando i marker da identificare diventano tre 6.16.

In questo caso si hanno comunque delle prestazioni adatte a un task da eseguire in tempo reale, il sistema elabora oltre i 10 frame per secondo, ma comunque inferiori ad ArUco.

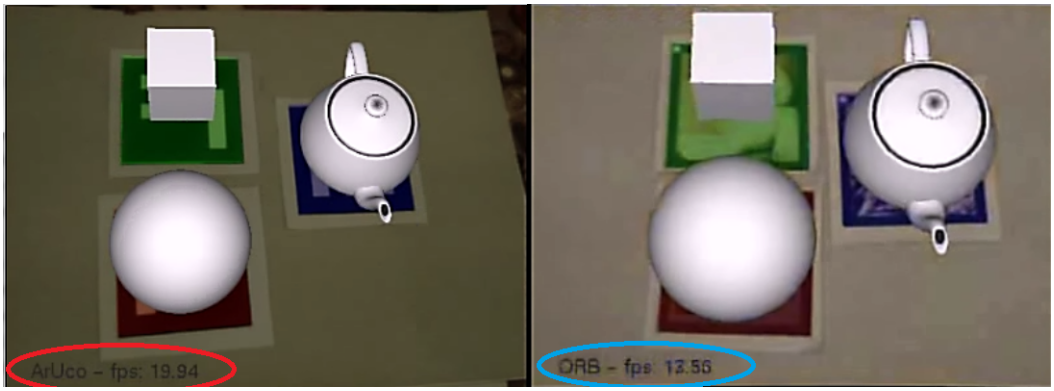


Fig. 6.16: Confronto delle prestazioni in termini di frame per secondo tra l'architettura di base ArUco e l'architettura estesa con algoritmo ORB, utilizzando tre marker nella scena.

6.7 Conclusioni sui test

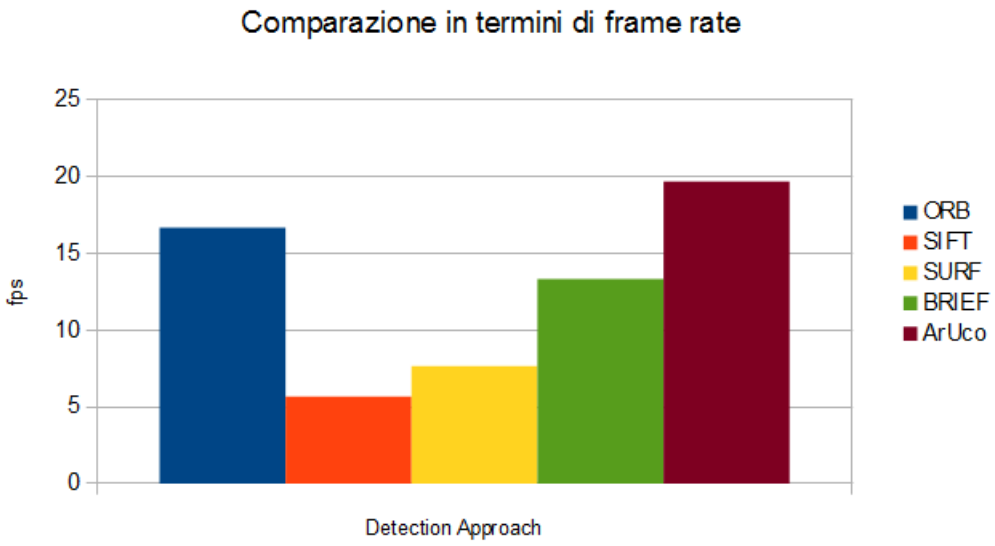


Fig. 6.17: Andamento medio del framerate dell'architettura inizializzata di volta in volta con gli algoritmi supportati. I dati si riferiscono al test effettuato utilizzando lo stesso video per ognuna delle configurazioni.

Con i test effettuati è stata dimostata l'applicabilità della procedura di detection che utilizza ORB in situazioni nelle quali è richiesto una responsività classica dei sistemi realtime. L'implementazione fornita pur non potendo eguagliare l'algoritmo di fiducial marker detection di ArUco, si avvicina sensibilmente alle sue prestazioni in condizioni classiche 6.17, ovvero con un unico marker presente all'interno della scena e continua ad avere prestazioni soddisfacenti aumentando il numero di marker presenti nella scena.

Capitolo 7

Conclusioni e sviluppi futuri

L'obiettivo della seguente tesi è stato quello di proporre e valutare un metodo per il *tracking* di marker con caratteristiche naturali all'interno di un flusso di immagini provenienti da una telecamera. Sono state utilizzate tecniche di Computer Vision e algoritmi noti in letteratura per l'individuazione di punti caratteristici all'interno di immagini. Tali algoritmi sono stati studiati e analizzati nelle prestazioni in termini di affidabilità e di velocità di calcolo per poter effettuare considerazioni riguardo l'approccio da utilizzare in quanto miglior compromesso tra velocità computazionale e potenza descrittiva. E' stata presentata un'architettura in grado di assolvere a tale compito, basata su un' infrastruttura già sviluppata nella libreria opensource ArUco, ampliata a tal proposito per poter ottenere un approccio utilizzabile in pratica in applicazioni di Realtà Aumentata.
Il lavoro è stato svolto organizzandolo logicamente in 6 capitoli più il presente:

Il **Capitolo** 1 ha fornito una definizione di Realtà Aumentata (AR) descrivendo e collocando al suo interno il problema fondamentale dell' Image Matching.

Il **Capitolo** 2 ha dato una descrizione dello stato dell'arte degli algoritmi per Image Matching off-line.

Nel **Capitolo** 3 è stato descritto lo stato dell'arte di altri algoritmi per Image Matching con caratteristiche tali da permettere calcoli veloci e l'impiego in applicazioni interattive.

Il **Capitolo** 4 ha fornito la descrizione di un tipico algoritmo di marker detection con marker fiduciali bianchi e neri e introdotto ArUco, una libreria opensource per la Realtà Aumentata.

Il **Capitolo** 5 ha presentato l'implementazione di un tracker di marker con caratterestiche naturali basato su ArUco.

Il **Capitolo** 6 ha descritto i test effettuati sul campo, allo scopo di fornire una valutazione delle prestazioni di alcuni tra gli algoritmi descritti nello stato dell'arte e impiegati nel processo di tracking implementato nel presente progetto.

7.1 Conclusioni

Il lavoro svolto in questa tesi si propone di fornire un meccanismo automatico di rilevamento di marker non usuali, come i classici marker fiduciali bianchi e neri, ma estende tale comportamento nei confronti di marker che hanno le caratteristiche generiche di immagini classiche.
Lo scopo finale di questo lavoro di fatti è stato quello di permettere il tracciamento di marker riducendo al minimo l'invasività che gli stessi hanno nella scena. Durante una ripresa l'occhio dell'utente è disturbato dalla presenza di oggetti bianchi e neri che non hanno un'armonia con il contensto nel quale sono inseriti 7.1(a), mentre immagini classiche vengono meglio mascherate e

rendono meno artificiale l'esperienza finale 7.1(b). Si pensi ad un'applicazione di Realtà Aumentata inserita in un contesto come quello di una mostra di quadri, nel quale è possibile, con gli strumenti forniti, individuare e riconoscere un'opera rispetto ad un' altra senza la necessità di dovervi applicare marker fiduciali, ma limitandosi ad applicare una cornice di colore scuro. I

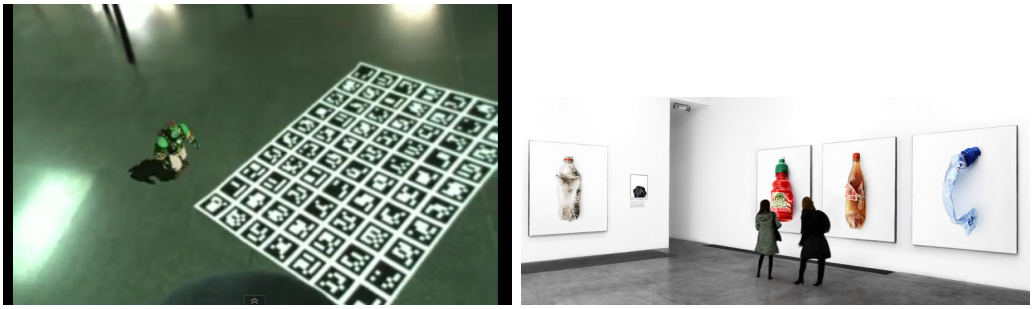


Fig. 7.1: Impatto dell'invasività dei markers nell' esperienza dell'utente

risultati dei test effettuati dimostrano l'usabilità degli strumenti forniti in applicazioni che necessitano di risposte in tempo reale e allo stesso tempo offrono lo spunto per alcune riflessioni sui punti critici della soluzione qui implementata. I difetti principali riguardano i malfunzionamenti dovuti alle condizioni dell'ambiente entro il quale avviene l'esperienza virtuale:

- in condizioni di scarsa illuminazione il rilevamento funziona male in quanto nel frame analizzato non viene riconosciuto nessun marker, la stessa cosa può capitare se i marker nella scena sono troppo vicini a una fonte di luce, quest'ultimo problema può essere attenuato producendo marker con materiali anti-riflettenti.
- un'altra causa di malfunzionamento riguarda l'occlusione dei bordi, infatti il marker occluso non viene rilevato in quanto non viene individuato il bordo entro il quale fare l'analisi dell'immagine.
- le stesse considerazioni valgono nel caso di ambienti estesi nei quali i marker possono essere distanti dalla videocamera e quindi difficilmente

rilevabili.

7.2 Sviluppi futuri

Sulla base dell'esperienza accumulata durante la presente tesi e considerando i risultati dei test effettuati e le considerazioni sopra esposte, è possibile individuare tra gli sviluppi futuri più interessanti del presente lavoro i seguenti punti:

- La riduzione dell'invasività, comunque presente nell'implementazione fornita e causata dai bordi del marker, è un punto sul quale si può apportare un miglioramento sensibile, raggiungendo una struttura di rilevamento completamente *markerless*, preoccupandosi comunque di trovare una tecnica per la quale l'area da analizzare non sia l'intero frame.
- Estendere il meccanismo fornito da ArUco dei *Boards* anche nel caso di marker naturali permette di ridurre problemi di illuminazione locale non ottimale o di occlusione parziale.
- La struttura presentata si presta all'estensione naturale che deriva dall'integrazione di nuovi algoritmi di image matching in OpenCV, semplicemente aggiungendo alla classe `NaturalFeaturesUtils` le nuove procedure.

Bibliografia

[1] Aruco a minimal library for augmented reality applications based on opencv. http://www.uco.es/investiga/grupos/ava/node/26/.

[2] Atan2. http://en.wikipedia.org/wiki/Homography.

[3] Atan2. http://en.wikipedia.org/wiki/Atan2.

[4] Descriptor matcher. http://opencv.willowgarage.com/documentation/cpp/features2d_common_interfaces_of_descriptor_matchers.html.

[5] Generic descriptor matcher. http://opencv.willowgarage.com/documentation/cpp/features2d_common_interfaces_of_generic_descriptor_matchers.html.

[6] Greedy algotithm. http://en.wikipedia.org/wiki/Greedy_algorithm.

[7] The industry's foundation for high performance graphics. http://www.opengl.org/.

[8] Opencv willowgarage's blog. http://opencv.willowgarage.com/wiki/.

[9] Features descriptor comparison report. http://computer-vision-talks.com/2011/08/feature-descriptor-comparison-report/, August 2011.

[10] Van Gool L. Bay H., Tuytelaars T. Surf: Speeded up robust features.

[11] Fua P. Calonder M., Lepetit V. BRIEF: Binary Robust Independent Elementary Features.

[12] Lowe D.G. Object recognition from local scale-invariant features. September 1999.

[13] Lowe D.G. Distinctive image features from scale-invariant keypoints. January 2004.

[14] Adrian Kaehler G.Bradski. *Learning Computer Vision with the OpenCV Library.* O'Reilly, 2008.

[15] Adrian Kaehler G.Bradski. *Learning Computer Vision with the OpenCV Library, pag. 316-364.* O'Reilly, 2008.

[16] Adrian Kaehler G.Bradski. *Learning Computer Vision with the OpenCV Library, pag 370-401.* O'Reilly, 2008.

[17] Adrian Kaehler G.Bradski. *Learning Computer Vision with the OpenCV Library, pag. 405-454.* O'Reilly, 2008.

[18] Adrian Kaehler G.Bradski. *Learning Computer Vision with the OpenCV Library, pag.144-148.* O'Reilly, 2008.

[19] Adrian Kaehler G.Bradski. *Learning Computer Vision with the OpenCV Library, pag.182.* O'Reilly, 2008.

[20] Christian Doppler Laboratory. Handheld Augmented Reality. `http://studierstube.icg.tu\discretionary{-}{}{}graz.ac.at/handheld_ar/index.php`, 2008.

[21] Mikolajczyk K. Detection of local features invariant to affine transformations. *Ph.D. Thesis*, 2002.

[22] Mikolajczyk K. and Schmid C. A Performance Evaluation of local descriptors. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 27, NO. 10,*, 2005.

[23] Rosin P.L. Measuring corner properties.

[24] Drummond T. Rosten E. Machine learning for high-speed corner detection.

[25] Konololige K. Bradsky G. Ruble E., Rabaud V. ORB: an efficient alternative to SIFT or SURF.

[26] Lindeberg T. Scale-Space theory. *Journal of Applied Statistics*, pages 224–270, January 1994.

[27] Fua P. Tola E., Lepetit V. A fast local descriptor for dense matching.

[28] Fua P. Tola E., Lepetit V. DAISY: An Efficient Dense Descriptor Applied to Wide-Baseline Stereo. 2009.

Appendice A

Matrice di trasformazione

A.1 Matrice di Rotazione

La rotazione di un oggetto può essere esprimibile come una moltiplicazione di un vettore di coordinate per un'appropriata matrice quadrata. Si tratta fondamentalmente di descrivere in maniera diversa la posizione di un punto usando un diverso sistema di coordinate, cioè quello della videocamera al posto di quello dell'oggetto a cui appartiene il punto. Per semplificare basta pensare che ruotare la videocamera (cioè il suo sistema di coordinate) di un certo angolo, equivale a ruotare l'oggetto che si sta riprendendo dello stesso angolo ma nella direzione opposta. La rotazione di un oggetto nelle tre dimensioni può essere scomposta in tre rotazioni bidimensionali attorno a ciascuno dei tre assi, dove l'asse di rotazione viene mantenuto costante. Per essere più chiari, questo significa che un oggetto viene ruotato in sequenza attorno agli assi x; y; z con angoli diversi per ciascun asse (rispettivamente ψ, ϕ, θ): così facendo viene ottenuta una matrice di rotazione R che è il prodotto delle tre matrici di rotazione associate a ciascun asse (siano $R_x(\psi); R_y(\phi); R_z(\theta)$)

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos_\psi & \sin_\psi \\ 0 & -\sin_\psi & \cos_\psi \end{bmatrix}$$

$$R_y(\phi) = \begin{bmatrix} \cos_\phi & 0 & -\sin_\phi \\ 0 & 1 & 0 \\ \sin_\phi & 0 & \cos_\phi \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos_\theta & \sin_\theta & 0 \\ -\sin_\theta & \cos_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

A.2 Vettore di Traslazione

Per quanto riguarda invece il vettore di traslazione, esso rappresenta semplicemente lo spostamento dal centro di un sistema di coordinate ad un altro: in pratica è l'offset (la distanza) dall'origine del sistema di coordinate dell'oggetto inquadrato al centro del sistema di coordinate della videocamera. Possiamo pertanto esprimerlo semplicemente come $T = origin_{object} - origin_{camera}$. Quindi un punto di un oggetto inquadrato di coordinate P_o avrà coordinate P_c nel sistema di riferimento della videocamera:

$$P_c = R(P_o - T) \tag{A.2}$$

Ogni singolo punto P_c inoltre andrà corretto tramite i valori contenuti nel vettore di distorsione, in modo da considerare anche le distorsioni radiale e tangenziale.

A.3 La matrice di trasformazione

Tutti i parametri trovati fino a questo punto (parametri intrinseci, di distorsione, di rotazione e di traslazione) vanno calcolati risolvendo un sistema di equazioni, prima di poterlo fare è però necessario mappare i punti dell'oggetto planare (la scacchiera) sul piano immagine. A questo scopo è necessaria una matrice di trasformazione detta matrice omografica (homography matrix) oppure matrice di proiezione prospettica. La procedura di mappare un punto da un piano bidimensionale ad un altro con coordinate e dimensioni diverse (nel nostro caso l'imager della camera) è detta omografia planare ed è esprimibile tramite moltiplicazione di matrici. Consideriamo i vettori Q e q contenenti le coordinate omogenee rispettivamente dell'oggetto reale e della sua proiezione:

$$Q = [XYZ1]^T$$

$$q = [xy1]^T \tag{A.3}$$

La moltiplicazione da eseguire è:

$$q = sHQ \tag{A.4}$$

Dove si è introdotto il fattore di scala s per il quale è definita l'omografia e che esprime il rapporto tra le dimensioni dell'oggetto reale Q e la sua proiezione q. H è la matrice omografica e a sua volta è composta da due parti: la matrice di trasformazione fisica che posiziona i punti dell'immagine sul piano e la matrice di proiezione costruita a partire dai parametri intrinseci della videocamera.

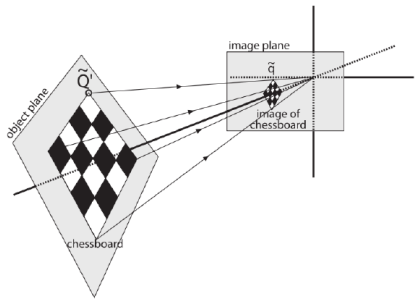


Fig. A.1: Schema delle operazioni descritte dall'omografia: mappatura dal piano reale al piano immagine.

La matrice di trasformazione fisica W è l'effetto della traslazione e della rotazione relative al piano reale che si sta osservando dal punto di vista del piano immagine: $W = [Rt]$ Sia invece M la matrice dei parametri intrinseci, sostituendo nella (A1):

$$q = sMWQ, doveM = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

Considerando che sia l'immagine di calibrazione, sia il fiducial di cui ci interessa la posizione e orientamento sono in realtà bidimensionali (immagini planari), possiamo semplificare imponendo Z = 0, così facendo, spezzando la matrice di rotazione R in tre colonne, la terza, che si riferisce all'asse z può essere eliminata:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM[r_1 r_2 r_3 t] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = sM[r_1 r_2 t] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{A.6}$$

La mappatura dei punti di un oggetto planare sul piano immagine viene descritta dalla matrice omografica 3X3 $H = sM[r1r2t]$:

$$q = sHQ \tag{A.7}$$

Queste equazioni vengono sfruttate per permettere alle procedure di OpenCV di calcolare la matrice H. Per farlo occorrono più immagini dell'oggetto a diverse angolazioni in modo da fornire diverse matrici di rotazione e vettori di traslazione, mentre i parametri intrinseci rimangono gli stessi. In totale, per ogni vista, devono essere trovate sei incognite: i tre angoli di rotazione e i tre offset di traslazione che si trovano nelle otto equazioni che derivano da ciascuna vista dell'oggetto, che consistono nella mappatura dei quattro vertici della scacchiera sul piano immagine (ogni vertice è descritto da una coppia di coordinate (x; y)):

$$p_{dst} = Hp_{src}, p_{src} = H^{-1}p_{dst}$$

$$p_{dst} = \begin{bmatrix} x_{dst} \\ y_{dst} \\ 1 \end{bmatrix}, p_{src} = \begin{bmatrix} x_{src} \\ y_{src} \\ 1 \end{bmatrix} \tag{A.8}$$

Non occorre che i parametri intrinseci siano noti a priori, infatti nella costruzione di H a partire da viste differenti, è possibile ricavare anche questi dati. Riassumendo, per ogni frame si hanno otto equazioni con le sei incognite estrinseche da risolvere: come vedremo, con un numero adeguato di

frame è possibile risolvere il sistema. I sistemi di equazioni A.1 e A.2 contengono i parametri necessari alla calibrazione della videocamera, e cioè i tre angoli specificati per la rotazione e i tre offset (uno per ciascun asse) della traslazione: in tutto sei parametri. A questi vanno aggiunti anche i quattro parametri intrinseci e i cinque parametri dovuti alla distorsione, portando il totale a quindici parametri da risolvere. Una singola vista dell'oggetto di calibrazione, da sola, permette di ricavare i cinque parametri del vettore di distorsione. Per quanto riguarda i parametri estrinseci (rotazione e traslazione) occorre sapere in che punto dello spazio si trova l'oggetto: ciò è ricavabile grazie proprio ai tre angoli di rotazione ed alle tre coordinate di traslazione, che cambiano per ogni punto di vista da cui viene ripreso l'oggetto. I parametri da risolvere sono quindi in tutto dieci per ogni vista. Ma quante viste occorrono per riuscire ad ottenere tutti i parametri? Come accennato in precedenza, l'oggetto usato per la calibrazione è costituito da una scacchiera o da una griglia di elementi regolari. Gli angoli formati tra i quadrati della scacchiera sono sempre uguali per ciascun punto di vista da cui si riprende l'oggetto, possono essere quindi usati come punto di riferimento costante per poter stabilire l'angolazione dell'oggetto. Supponiamo di avere a disposizione K viste di una scacchiera in cui sono visibili N angoli:

- K viste di un oggetto forniscono 2NK vincoli (NK vincoli per ciascun asse).
- In totale si hanno 4 parametri intrinseci e 6K parametri estrinseci.
- Per poter risolvere il sistema occorre un numero sufficiente di vincoli, cioè: $2NK \geq 6K + 4$.

Senza scendere troppo nei dettagli, si trova che bastano due viste per poter risolvere il sistema e trovare tutti i parametri necessari alla calibrazione della videocamera, chiaramente un numero maggiore di viste renderà il risultato

della calibrazione più affidabile e meno soggetto al rumore.